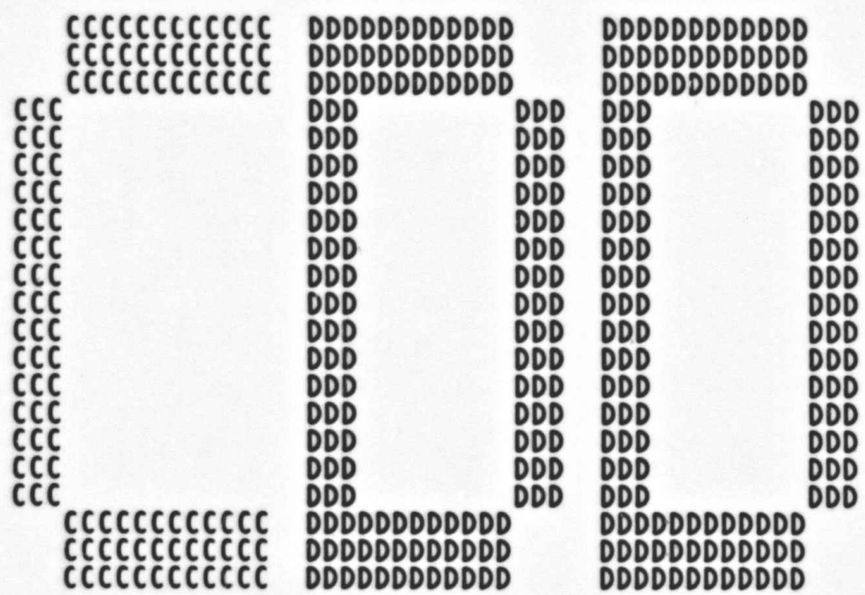

CCCCCCCCCCCC DDDDDDDDDDDD DDDDDDDDDDDD
CCCCCCCCCCCC DDDDDDDDDDDD DDDDDDDDDDDD
CCCCCCCCCCCC DDDDDDDDDDDD DDDDDDDDDDDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCCCCCCCCCCC DDDDDDDDDDDD DDDDDDDDDDDD
CCCCCCCCCCCC DDDDDDDDDDDD DDDDDDDDDDDD
CCCCCCCCCCCC DDDDDDDDDDDD DDDDDDDDDDDD

```
  CCCCCCCC  DDDDDDD    DDDDDDD    LL          IIIIII   BBBBBBBB
  CCCCCCCC  DDDDDDDD   DDDDDDDD   LL          IIIIII   BBBBBBBB
CC          DD      DD DD      DD LL            II     BB      BB
CC          DD      DD DD      DD LL            II     BB      BB
CC          DD      DD DD      DD LL            II     BB      BB
CC          DD      DD DD      DD L!            II     BB      BB
CC          DD      DD DD      DD LL            II     BBBBBBBB
CC          DD      DD DD      DD LL            II     BBBBBBBB
CC          DD      DD DD      DD LL            II     BB      BB
CC          DD      DD DD      DD LL            II     BB      BB
CC          DD      DD DD      DD LL            II     BB      BB
CC          DD      DD DD      DD LL            II     BB      BB
  CCCCCCCC  DDDDDDDD   DDDDDDDD   LLLLLLLLLL  IIIIII   BBBBBBBB
  CCCCCCCC  DDDDDDD    DDDDDDD    LLLLLLLLLL  IIIIII   BBBBBBBB
```

```
BBBBBBBB     333333     222222
BBBBBBBB     333333     222222
BB      BB 33      33 22      22
BB      BB 33      33 22      22
BB      BB         33         22
BB      BB         33         22
BBBBBBBB       33           22
BBBBBBBB       33           22
BB      BB       33       22
BB      BB       33       22
BB      BB 33    33     22
BB      BB 33    33     22
BBBBBBBB     333333   2222222222
BBBBBBBB     333333   2222222222
```

```
!++
!  TITLE:  CDDLIB                          CDD Bliss Library
!
!  VERSION:  'V04-000'
!
!  FACILITY:  Common Data Dictionary
!
!  ABSTRACT:
!
!       This module is the library file used for compiling all other
!       modules in the CDD facility.
!
!  ENVIRONMENT:
!
!  AUTHOR:  Jeff East,  22-Jan-80
!
!  MODIFIED BY:
!
!       7-May-81 (JAE)  Added $IO_SYNC macro.
!--
    %TITLE      'CDD Bliss Library'
```

```
    %SBTTL      'STRUCTURE DEFINITIONS'
!++
!
!       STRUCTURE DEFINITIONS
!
!--


!+
!       On-disk Pointer Structure
!
!               23                  6       0
!                +------------------+-------+ +0
!                |                  |       |
!                |  Logical Page    | Line  |      DPTR$V_LINE
!                |     number       |number |      DPTR$V_PAGE
!                +------------------+-------+ +3
!-

LITERAL
    DPTR$S_BLOCK_LENGTH = 3;

MACRO
    $DPTR = BLOCK[DPTR$S_BLOCK_LENGTH,BYTE] FIELD (DPTR$Z_FIELDS)
    %;

FIELD  DPTR$Z_FIELDS =
    SET
        DPTR$V_VALUE            = [0, 0, 24, 0],
        DPTR$V_LINE             = [0, 0, 7, 0],
        DPTR$V_PAGE             = [0, 7, 17, 0]
    TES;
```

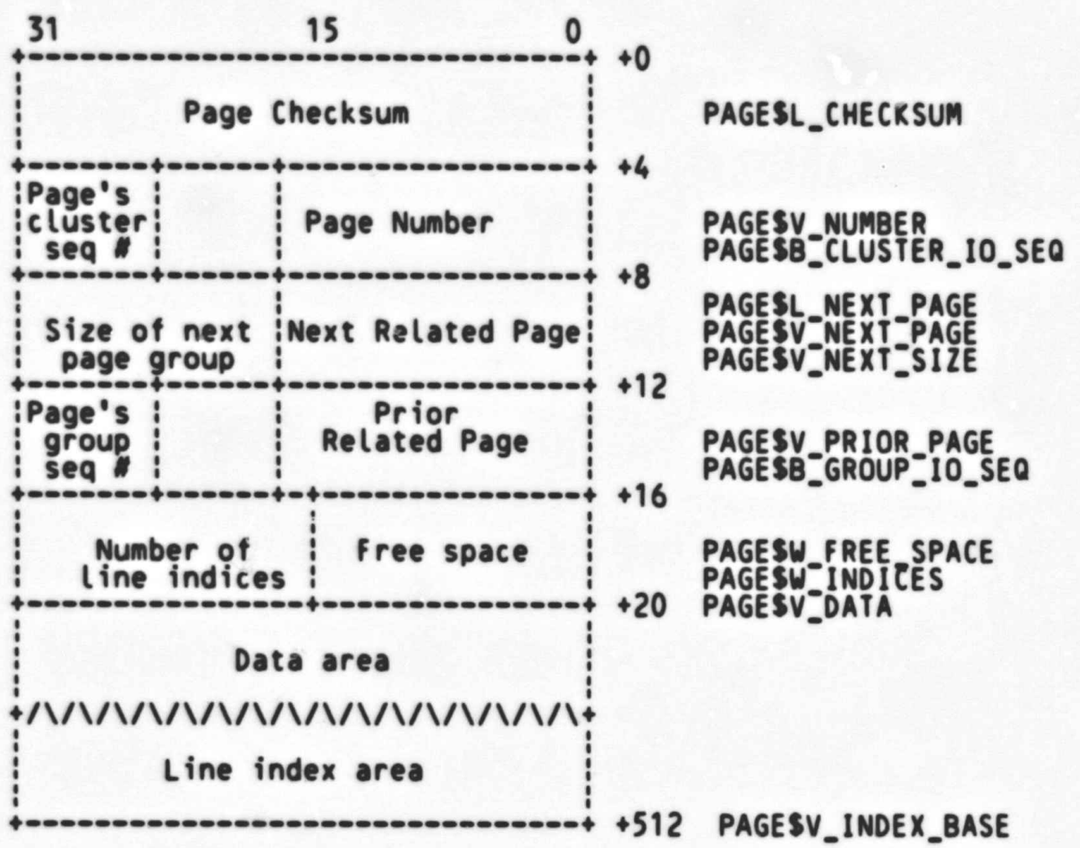
```
!+
!       In-core Disk Pointer Structure
!
!
!      31       23                  6      0
!       +--------+-----------------+------+ +0
!       |Internal|                 |      |      DKEY$V_LINE
!       |  file  |  Logical Page   | Line |      DKEY$V_PAGE
!       | number |     number      |number|      DKEY$B_FILE
!       +--------+-----------------+------+ +4
!
!       When a disk pointer is being passed around routines, it
!       is passed as a Disk Key (DKEY), rather than just a disk
!       pointer.
!
!       Disk Keys include all the information of a disk pointer,
!       but also include a pointer to their file's FCB.
!-

LITERAL
    DKEY$S_BLOCK_LENGTH = 4;

MACRO
    $DKEY = BLOCK[DKEY$S_BLOCK_LENGTH,BYTE] FIELD (DKEY$Z_FIELDS)
    %;

FIELD  DKEY$Z_FIELDS =
    SET
        DKEY$V_LINE             = [0, 0, 7, 0],
        DKEY$V_PAGE             = [0, 7, 17, 0],
        DKEY$B_FILE             = [0, 24, 8, 0]
    TES;
```

```
!+
!       In-core Page Number Structure
!
!
!      31       23                          0
!       +--------+--------------------------+ +0
!       |        |                          |
!       |  File  |       Page number        |      PKEY$V_PAGE
!       | number |                          |      PKEY$B_FILE
!       +--------+--------------------------+ +4
!
!       Dictionary page numbers that are passed around the hash table
!       use the Page Key (PKEY) structure to provide both the page
!       and file numbers.
!-

LITERAL
    PKEY$S_BLOCK_LENGTH = 4;

MACRO
    $PKEY = BLOCK[PKEY$S_BLOCK_LENGTH,BYTE] FIELD (PKEY$Z_FIELDS)
    %;

FIELD  PKEY$Z_FIELDS =
    SET
        PKEY$V_PAGE             = [0, 0, 24, 0],
        PKEY$B_FILE             = [0, 24, 8, 0]
    TES;
```

```
!+
!       Line Index Format
!
!
!        31              15              0
!                        +---------------+ +0
!                        |               |
!                        | Offset to blk |      LINE$W_BLOCK_OFFSET
!                        |from page base |
!                        +---------------+ +2
!
!       Each block on a dictionary page is pointed to by a line index
!       on that page.  Disk pointers (DPTR) between the various blocks
!       actually point to the block's line index.  The line index is
!       then used to locate the physical dictionary block.
!
!       The PAGE$INDEX macro allows the programmer to access a line
!       index while it resides on a dictionary page.  It also uses the
!       LINE$Z_FIELDS to access the individual fields within a line index.
!-

LITERAL
    LINE$S_BLOCK_LENGTH = 2;

MACRO
    $LINE = BLOCK[LINE$S_BLOCK_LENGTH,BYTE] FIELD (LINE$Z_FIELDS)
    %;

FIELD  LINE$Z_FIELDS =
    SET
        LINE$W_BLOCK_OFFSET     = [0, 0, 16, 0]
    TES;
```

!+

Page Format



Each page in the dictionary file starts with a page header. The next and prior related page pointers are used for linking page groups. The following types of page groups exist:

1) Lockable page group.
   Each named entity (and history list) owns exactly one lockable page group. These groups contain all unamed offspring of the name entiry (or history list).

   A page group may only be accessed through its portal page. A group's portal page is the page on which the named entity (or history list head) resides. If a group's portal page is locked, then no pages in the group may be accessed.

The PAGE$INDEX structure is used to access a line index entry on a page.

A page whose checksum is zero is a locked page, and indicates that

```
!       the sub-tree below it is incomplete and in a transient state.
!       Such pages may never be read.
!
!       Each page has two page sequence numbers.  The cluster sequence
!       number must be the same for all pages in the cluster.  The group
!       sequence number must be the same for all pages in an I/O group.
!
!       The cluster I/O sequence number is bumped whenever more than one
!       group in the cluster is written.  The group I/O sequence number
!       is bumped whenever the group is written.  These sequence numbers
!       enable the detection of incomplete cluster unstage operations.
!-

LITERAL
    PAGE$S_BLOCK_LENGTH = 512;

MACRO
    $PAGE = BLOCK[PAGE$S_BLOCK_LENGTH,BYTE] FIELD (PAGE$Z_FIELDS)
    %;

FIELD PAGE$Z_FIELDS =
    SET
        PAGE$L_CHECKSUM         = [0, 0, 32, 0],
        PAGE$V_NUMBER           = [4, 0, 17, 0],
        PAGE$B_CLUSTER_IO_SEQ   = [7, 0, 8, 0],
        PAGE$L_NEXT_PAGE        = [8, 0, 32, 0],
        PAGE$V_NEXT_PAGE        = [8, 0, 17, 0],
        PAGE$V_NEXT_SIZE        = [8, 17, 15, 0],
        PAGE$V_PRIOR_PAGE       = [12, 0, 17, 0],
        PAGE$B_GROUP_IO_SEQ     = [15, 0, 8, 0],
        PAGE$W_FREE_SPACE       = [16, 0, 16, 0],
        PAGE$W_INDICES          = [18, 0, 16, 0],
        PAGE$V_DATA             = [20, 0, 0, 0],
        PAGE$V_INDEX_BASE       = [512, 0, 0, 0]
    TES;

LITERAL
    PAGE$K_BASE         = BLOCK[0, PAGE$V_DATA;
                                PAGE$S_BLOCK_LENGTH, BYTE],
    PAGE$K_FREE_SPACE   = PAGE$S_BLOCK_LENGTH - PAGE$K_BASE,
    PAGE$S_INDEX_BASE   = BLOCK[0, PAGE$V_INDEX_BASE;
                                PAGE$S_BLOCK_LENGTH, BYTE];

STRUCTURE
    PAGE$INDEX[I, O, P, S, E] =
        (PAGE$INDEX+PAGE$S_INDEX_BASE+O-((I)*LINE$S_BLOCK_LENGTH)) <P,S,E>;
```

```
!+
!       Dictionary Header Block
!
!
!               23      15                      0
!                +------------------------+ +0
!                |                        |
!                |  Pointer to top of     |      DICT$V_CDD$TOP
!                | dictionary sub-tree    |
!                +------------------------+ +3
!                |                        |
!                |  Pointer to last       |      DICT$V_LAST_BITMAP
!                | free page bit-map      |
!                +------------------------+ +6
!
!       Each dictionary uses page 1 as its dictionary header page.
!       The dictionary header block immediately follows the page
!       header on page 1.
!
!       DICT$V_CDD$TOP points to the root of the tree/sub-tree contained
!       in the dictionary file.
!
!       DICT$V_LAST_BITMAP points to the last free page bit-map entry.
!-

LITERAL
    DICT$S_BLOCK_LENGTH = 6 + PAGE$K_BASE;

MACRO
    $DICT = BLOCK[DICT$S_BLOCK_LENGTH,BYTE] FIELD (PAGE$Z_FIELDS,
                                                   DICT$Z_FIELDS)
    %;

FIELD DICT$Z_FIELDS =
    SET
        DICT$V_CDD$TOP          = [0+PAGE$K_BASE, 0, 24, 0],
        DICT$V_LAST_BITMAP      = [3+PAGE$K_BASE, 0, 24, 0]
    TES;
```

```
!+
!       Universal Block Header
!
!      31       24       15                0
!       +----------------+-------+--------+ +0
!       |                |       |        |      BLK$B_TYPE
!       |  Block Length  | Block | Block  |      BLK$B_FLAGS
!       |                | Flags | Type   |      BLK$W_LENGTH
!       +----------------+-------+--------+ +4
!
!       This block header is used in every non-fixed block on the
!       dictionary pages.  Fixed blocks (one which always occur
!       in the same place whenever they exist) do not have this
!       block header.
!-

LITERAL
    BLK$S_BLOCK_LENGTH  = 4;

MACRO
    $BLK = BLOCK[BLK$S_BLOCK_LENGTH,BYTE] FIELD (ACL$Z_FIELDS,
                                                 ACLC$Z_FIELDS,
                                                 ATT$Z_FIELDS,
                                                 BLK$Z_FIELDS,
                                                 ELST$Z_FIELDS,
                                                 ENT$Z_FIELDS,
                                                 FPCB$Z_FIELDS,
                                                 LIST$Z_FIELDS,
                                                 LST$Z_FIELDS,
                                                 NAME$Z_FIELDS,
                                                 NAT$Z_FIELDS,
                                                 NNAM$Z_FIELDS,
                                                 NODE$Z_FIELDS,
                                                 SEG$Z_FIELDS,
                                                 SLST$Z_FIELDS,
                                                 SSA$Z_FIELDS,
                                                 STR$Z_FIELDS,
                                                 TEXT$Z_FIELDS)
    %;

FIELD BLK$Z_FIELDS =
    SET
        BLK$B_TYPE              = [0, 0, 8, 0],
        BLK$B_FLAGS             = [1, 0, 8, 0],
        BLK$W_LENGTH            = [2, 0, 16, 0]
    TES;

LITERAL
    BLK$K_FLAGS         = BLOCK[0, BLK$B_TYPE; ,BYTE],
    BLK$K_BASE          = BLK$S_BLOCK_LENGTH;

!+
```

```
!       Block types
!
!       The ordering and clustering of these block types is significant.
!
!       All block types must be contiguous without any holes and
!       bounded by the symbols BLK$K_TYPE_FIRST and BLK$K_TYPE_LAST.
!
!       All node and NAME block types must be contiguous and bounded
!       by the symbols BLK$K_TYPE_NODE_FIRST and BLK$K_TYPE_NODE_LAST.
!
!       To add more node or NAME block types, insert them at the front
!       of the table.
!
!       To add any other block type, append them to the end of the table.
!
!       CHANGING THE VALUES OF ANY OF THE ACTUAL BLOCK TYPES WILL INVALIDATE
!       EXISTING DICTIONARIES.
!-

LITERAL
    BLK$K_TYPE_FIRST                    = 101,  ! Lowest block type

        BLK$K_TYPE_NODE_FIRST           = 101,  ! First node or NAM block

            BLK$K_TYPE_DIR_NAM          = 101,  ! Directory NAM block
            BLK$K_TYPE_FIL_NAM          = 102,  ! File NAM block
            BLK$K_TYPE_TERM_NAM         = 103,  ! Terminal NAM block
            BLK$K_TYPE_DIR_NODE         = 104,  ! Directory node block
            BLK$K_TYPE_TERM_NODE        = 105,  ! Terminal node block

        BLK$K_TYPE_NODE_LAST            = 105,  ! Last node or NAM block

        BLK$K_TYPE_BITMAP               = 106,  ! Free page bitmap
        BLK$K_TYPE_ENTITY_ATT           = 107,  ! Entity attribute block
        BLK$K_TYPE_ENTITY_LIST          = 108,  ! Entity list block
        BLK$K_TYPE_ENTITY_LIST_ATT      = 109,  ! Entity list attribute block
        BLK$K_TYPE_NULL_ATT             = 110,  ! Null attribute block
        BLK$K_TYPE_NUM_ATT              = 111,  ! Numeric attribute block
        BLK$K_TYPE_SHORT_ATT            = 112,  ! Short string attribute block
        BLK$K_TYPE_STRING_ATT           = 113,  ! String attribute block
        BLK$K_TYPE_STRING_LIST          = 115,  ! String list block
        BLK$K_TYPE_STRING_LIST_ATT      = 116,  ! String list attribute block
        BLK$K_TYPE_STRING_SEG           = 117,  ! String segment block
        BLK$K_TYPE_TEXT                 = 118,  ! Text block
        BLK$K_TYPE_ACL                  = 119,  ! Access control list entry
        BLK$K_TYPE_ACLC                 = 120,  ! Access control list criterion
    BLK$K_TYPE_LAST                     = 120;  ! Highest block type
```

```
!+
!       Free Page Control Block (FPCB)
!       Free Page Bit Map (FPBM)
!
!
!                +------------------------+ +0
!                |                        |
!                |  Disk address of prior |      FPCB$V_PRIOR
!                |        FBCB            |
!       +--------+------+-----------------+ +3
!       |                |                 |
!       |   Number of    |   First page    |      FPCB$V_FIRST_PAGE
!       | pages mapped   |     mapped      |      FPCB$V_PAGE_COUNT
!       +----------------+-----------------+ +7
!                        |                 |
!                        |  Blocks not     |      FPCB$W_EXCESS_BLOCKS
!                        |  used here      |
!                        +-----------------+ +9   FPCB$V_BITMAP
!
!       Free pages in the dictionary file are controlled by the Free Page
!       Control Blocks (FPCB) and the Free Page Bit Map (FPBM).  Each
!       free page in the file has its corresponding bit set on.
!
!       The disk's cluster size may make it impossible to use all of the
!       allocated blocks.  If so, the un-used blocks are tallied in
!       FPCB$W_EXCESS_BLOCKS, and will be used when the file is next
!       extended.
!
!       The FPCBs are chained as the dictionary file grows in size.
!-

LITERAL
    FPCB$S_BLOCK_LENGTH = BLK$K_BASE + 9;

STRUCTURE
    FPCB[O, P, S, E; BLOCKS] = [FPCB$S_BLOCK_LENGTH+(BLOCKS+7)/8]
                               (FPCB+O)<P,S,E>;

FIELD  FPCB$Z_FIELDS =
    SET
        FPCB$V_PRIOR            = [0+BLK$K_BASE, 0, 24, 0],
        FPCB$V_FIRST_PAGE       = [3+BLK$K_BASE, 0, 17, 0],
        FPCB$V_PAGE_COUNT       = [3+BLK$K_BASE, 17, 15, 0],
        FPCB$W_EXCESS_BLOCKS    = [7+BLK$K_BASE, 0, 16, 0],
        FPCB$V_BITMAP           = [9+BLK$K_BASE, 0, 0, 0]
    TES;

LITERAL
    FPCB$K_BASE   = FPCB$S_BLOCK_LENGTH,
    FPCB$S_BITMAP = FPCB[0, FPCB$V_BITMAP];

STRUCTURE
    FPBM[I] = (FPBM+FPCB$S_BITMAP)<I-1, 1, 0>;
```

```
!+
!       Name Block
!
!
!        31      24        15                0
!                 +-------------------------+ +0
!                 |                         |
!                 |  Pointer to next NAME   |      NAME$V_SIBLING
!                 |       (sibling)         |
!                 +-------------------------+ +3
!                 |                         |
!                 |  Pointer to name block  |      NAME$V_NAME
!                 |                         |
!                 +-------+-----------------+ +6
!                 |                         |
!                 |   Pointer to node       |
!                 |  block or forwarding    |      NAME$V_NODE or NAME$V_FILE
!                 |       file name         |
!                 +-------------------------+ +9
!
!       There are three types of NAM Blocks, Directory, File, and Terminal.
!
!       Directory NAM Block is made up of a Block Header + NAM Block.
!
!       File NAM Block is made up of a Block Header + NAM Block.
!
!       Terminal NAM Block is made up of a Block Header + NAM Block + Terminal
!       NAM Block.
!-

LITERAL
    NAME$S_BLOCK_LENGTH = 9 + BLK$K_BASE;

MACRO
    $NAME = BLOCK[NAME$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
                                                   NAME$Z_FIELDS)
    %;

FIELD NAME$Z_FIELDS =
    SET
        NAME$V_SIBLING          = [0+BLK$K_BASE, 0, 24, 0],
        NAME$V_NAME             = [3+BLK$K_BASE, 0, 24, 0],
        NAME$V_NODE             = [6+BLK$K_BASE, 0, 24, 0],
        NAME$V_FILE             = [6+BLK$K_BASE, 0, 24, 0]
    TES;

LITERAL
    NAME$K_BASE         = NAME$S_BLOCK_LENGTH;
```

```
!+
!       Node Nam Block
!
!
!        31      24        15                0
!                 +-----------------------+ +0
!                 |                       |
!                 |  Pointer to resource  |      NNAM$V_RESOURCE
!                 |       name block      |
!                 +-----------------------+ +3
!
!       Node NAM Block is made up of a Block Header + NAM Block + Node
!       NAM Block.
!-

LITERAL
    NNAM$S_BLOCK_LENGTH = 3 + NAME$K_BASE;

MACRO
    $NNAM = BLOCK[NNAM$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
                                                   NAME$Z_FIELDS,
                                                   NNAM$Z_FIELDS)
    %;

FIELD NNAM$Z_FIELDS =
    SET
        NNAM$V_RESOURCE = [0+NAME$K_BASE, 0, 24, 0]
    TES;

LITERAL
    NNAM$K_BASE         = NNAM$S_BLOCK_LENGTH;
```

```
!+
!       Node Block
!
!        31      24        15                0
!         +-----------------------------------+ +0
!         |  Pointer to name block            |      NODE$V_NAME
!         +-----------------------------------+ +3
!         |  Pointer to history               |      NODE$V_HISTORY
!         |       list head                   |
!         +-----------------------------------+ +6
!         |  Pointer to protection            |      NODE$V_PROTECTION
!         |       list head                   |
!         +-------+---------------------------+ +9
!         |  Pointer to parent                |      NODE$V_PARENT
!         |       node                        |
!         +-----------------------------------+ +12
!         |  Pointer to resource              |      NODE$V_RESOURCE
!         +-----------------------------------+ +15
!         |       Pointer to                  |      NODE$V_LIST
!         |        list head                  |
!         +-----------------------------------+ +18
!                 |       CDD         |             NODE$W_VERSION
!                 |     version       |
!                 +-------------------+ +20
!
!       There are two types of Node Blocks Directory and Terminal.
!
!       Directory Node Block is made up of a Block Header + Node Block
!
!       Terminal Node Block is made up of a Block Header + Node Block
!-

LITERAL
    NODE$S_BLOCK_LENGTH = 20+ BLK$K_BASE;

MACRO
    $NODE = BLOCK[NODE$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
                                                   NODE$Z_FIELDS)
    %;

FIELD NODE$Z_FIELDS =
    SET
        NODE$V_ORDERED          = [1, 0, 1, 0],
        NODE$V_NAME             = [0+BLK$K_BASE, 0, 24, 0],
        NODE$V_HISTORY          = [3+BLK$K_BASE, 0, 24, 0],
```

```
        NODE$V_PROTECTION       = [6+BLK$K_BASE, 0, 24, 0],
        NODE$V_PARENT           = [9+BLK$K_BASE, 0, 24, 0],
        NODE$V_RESOURCE         = [12+BLK$K_BASE, 0, 24, 0],
        NODE$V_LIST             = [15+BLK$K_BASE, 0, 24, 0],
        NODE$W_VERSION          = [18+BLK$K_BASE, 0, 16, 0]
    TES;

LITERAL
    NODE$K_BASE         = NODE$S_BLOCK_LENGTH;

!+
!       The following flag occurs in the BLK$B_FLAGS field of
!       a directory node.
!-

LITERAL
    NODE$M_ORDERED      = 1^1 - 1^0;    ! List elements are sorted by name
```

```
!+
!       Common Attribute Block
!
!        31     24       15              0
!       +----------------+---------------+ +0
!       |                                |
!       |         Attribute name         |      ATT$L_NAME
!       |                                |
!       +----------------+---------------+ +4
!       |                                |
!       |    Pointer to next sibling     |      ATT$V_SIBLING
!       |                                |
!       +--------------------------------+ +7
!
!       Each attribute has the common attribute block immediately following
!       the universal block header.
!
!       There are six types of Attribute Blocks Entity Attribute Block,
!       List Attribute Block, Numeric Attribute Block, Null Attribute Block,
!       Short String Attribute Block, and String Attribute Block.
!
!       Entity Attribute Block is made up of a Block Header + Attribute Block
!       + Entity Attribute Block.
!
!       List Attribute Block is made up of a Block Header + Attribute Block
!       + List Attribute Block.
!
!       Numeric Attribute Block is made up of a Block Header + Attribute Block
!       + Numeric Attribute BLock.
!
!       Null Attribute Block is made up of a Block Header + Attribute Block.
!
!       Short String Attribute Block is made up of a Block Header + Attribute
!       Block + Short String Attribute.
!
!       String Attribute Block is made up of a Block Header + Attribute Block
!       + String Attribute Block.
!-

LITERAL
    ATT$S_BLOCK_LENGTH = 7 + BLK$K_BASE;

MACRO
    $ATT = BLOCK[ATT$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
                                                 BLK$Z_FIELDS)
    %;

FIELD ATT$Z_FIELDS =
    SET
        ATT$L_NAME              = [0+BLK$K_BASE, 0, 32, 0],
        ATT$V_SIBLING           = [4+BLK$K_BASE, 0, 24, 0]
    TES;

LITERAL
```

```
    ATT$K_BASE          = ATT$S_BLOCK_LENGTH;
```

```
!+
!       Entity Attribute Block
!
!
!        31      24       15                0
!                 +-----------------------+ +0
!                 |                       |
!                 |  Pointer to first     |      ENT$V_FIRST_ATT
!                 |     attribute         |
!                 +-----------------------+ +3
!
!       Entity Attribute Block is made up of a Block Header + Attribute Block
!       + Entity Attribute Block.
!-

LITERAL
    ENT$S_BLOCK_LENGTH  = 3 + ATT$K_BASE;

MACRO
    $ENT = BLOCK[ENT$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
                                                 BLK$Z_FIELDS,
                                                 ENT$Z_FIELDS)
    %;

FIELD ENT$Z_FIELDS =
    SET
        ENT$V_FIRST_ATT         = [0+ATT$K_BASE, 0, 24, 0]
    TES;

LITERAL
    ENT$K_BASE          = ENT$S_BLOCK_LENGTH;
```

```
!+
!       List Attribute Block
!
!
!        31      24       15                0
!                 +-------------------------+ +0
!                 |                         |
!                 |  Pointer to first       |      LIST$V_FIRST_LST
!                 |    lst segment          |
!                 +-------+-----------------+ +3
!                         |                 |
!                         | Total number    |      LIST$W_CELL_COUNT
!                         |    of cells     |
!                         +-----------------+ +5
!
!       List Attribute Block is made up of a Block Header + Attribute Block
!       + List Attribute Block.
!-

LITERAL
    LIST$S_BLOCK_LENGTH = 5 + ATT$K_BASE;

MACRO
    $LIST = BLOCK[LIST$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
                                                   BLK$Z_FIELDS,
                                                   LIST$Z_FIELDS)
    %;

FIELD LIST$Z_FIELDS =
    SET
        LIST$V_FIRST_LST        = [0+ATT$K_BASE, 0, 24, 0],
        LIST$W_CELL_COUNT       = [3+ATT$K_BASE, 0, 16, 0]
    TES;

LITERAL
    LIST$K_BASE         = LIST$S_BLOCK_LENGTH;
```

```
!+
!       Numeric Attribute Block
!
!
!       31       24       15                0
!       +---------------------------------+ +0
!       |                                 |
!       |     Numeric Attribute Value     |      NAT$L_VALUE
!       |                                 |
!       +---------------------------------+ +4
!
!       Numeric Attribute Block is made up of a Block Header + Attribute Block
!       + Numeric Attribute BLock.
!
!       Numeric attribute block contains the
!       value of the numeric attribute.
!-

LITERAL
    NAT$S_BLOCK_LENGTH  = 4 + ATT$K_BASE;

MACRO
    $NAT = BLOCK[NAT$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
                                                 BLK$Z_FIELDS,
                                                 NAT$Z_FIELDS)
    %;

FIELD NAT$Z_FIELDS =
    SET
        NAT$L_VALUE             = [0+ATT$K_BASE,0,32,0]
    TES;

LITERAL
    NAT$K_BASE                  = NAT$S_BLOCK_LENGTH;
```

```
!+
!       Short String Attribute Block
!
!       31                              0
!                                +---------+ +0   SSA$T_STRING
!                                |         |
!                                | String  |
!                                /         \
!                                \         /
!                                |         |
!                                +---------+ +n
!
!       Short String Attribute Block is made up of a Block Header + Attribute
!       Block + Short String Attribute.
!
!       Strings whose total length is between 0 and 255 bytes are usually
!       stored using the short string attribute block.  If a string is
!       too long to be stored in an SSA, then it is stored using a normal
!       string attribute block (STR).
!-

LITERAL
    SSA$S_BLOCK_LENGTH  = 0 + ATT$K_BASE;

MACRO
    $SSA = BLOCK[SSA$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
                                                 BLK$Z_FIELDS,
                                                 SSA$Z_FIELDS)
    %;

FIELD SSA$Z_FIELDS =
    SET
        SSA$T_STRING            = [0+ATT$K_BASE, 0, 0, 0]
    TES;

LITERAL
    SSA$K_BASE          = SSA$S_BLOCK_LENGTH;
```

```
!+
!       String Attribute Block
!
!
!        31     24       15                0
!         +-------------------------+ +0
!         |                         |
!         |   Pointer to first      |      STR$V_STRING
!         |    string segment       |
!         +--------+----------------+ +3
!                  |                |
!                  |  Total string  |      STR$W_LENGTH
!                  |     length     |
!                  +----------------+ +5
!
!       String Attribute Block is made up of a Block Header + Attribute Block
!       + String Attribute Block.
!
!       The string is broken into one or more segments.  The string
!       segments are stored in SEG blocks.
!-

LITERAL
    STR$S_BLOCK_LENGTH  = 5 + ATT$K_BASE;

MACRO
    $STR = BLOCK[STR$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
                                                 BLK$Z_FIELDS,
                                                 STR$Z_FIELDS)
    %;

FIELD STR$Z_FIELDS =
    SET
        STR$V_STRING            = [0+ATT$K_BASE, 0, 24, 0],
        STR$W_LENGTH            = [3+ATT$K_BASE, 0, 16, 0]
    TES;

LITERAL
    STR$K_BASE          = STR$S_BLOCK_LENGTH;
```

```
!+
!       LST Segment Block
!
!
!        31      24       15                0
!                 +-------------------------+ +0
!                 |                         |
!                 |  Pointer to next lst    |      LST$V_NEXT_LST
!                 |                         |
!                 +--------+----------------+ +3
!                          |                |
!                          |Number of cells |      LST$W_CELL_COUNT
!                          |      here      |
!                          |                |
!                          +----------------+ +5   LST$T_CELLS
!
!       There are two types of LST Segment Blocks Entity List Block and
!       String List Attribute Block.
!
!       Entity List Block is made up of a Block Header + LST Block +
!       Entity List Block.
!
!       String List Attribute Block is made up of a Block Header + LST
!       Block + String List Attribute Block.
!-

LITERAL
    LST$S_BLOCK_LENGTH  = 5 + BLK$K_BASE;

MACRO
    $LST = BLOCK[LST$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
                                                 LST$Z_FIELDS)
    %;


FIELD LST$Z_FIELDS =
    SET
        LST$V_NEXT_LST          = [0 + BLK$K_BASE, 0, 24, 0],
        LST$W_CELL_COUNT        = [3 + BLK$K_BASE, 0, 16, 0],
        LST$T_CELLS             = [5 + BLK$K_BASE, 0, 0, 0]
    TES;

LITERAL
    LST$K_BASE          = LST$S_BLOCK_LENGTH;
```

```
!+
!       Entity List Block
!
!
!        31      24        15                    0
!                 +-------------------------+ +0
!                 |                         |
!                 |   Pointer to first      |      ELST$V_ATT
!                 |       attribute         |
!                 |                         |
!                 +-------------------------+ +3
!
!       Entity List Block is made up of a Block Header + LST Block +
!       Entity List Block.
!-

LITERAL
    ELST$S_BLOCK_LENGTH = 3;

MACRO
    $ELST = BLOCK[ELST$S_BLOCK_LENGTH,BYTE] FIELD (ELST$Z_FIELDS)
    %;


FIELD ELST$Z_FIELDS =
    SET
        ELST$V_ATT      = [0, 0, 24, 0]
    TES;

LITERAL
    ELST$K_BASE          = ELST$S_BLOCK_LENGTH;


STRUCTURE
    ELST$ELM[I] =
        (ELST$ELM + LST$K_BASE + (I * ELST$S_BLOCK_LENGTH));
```

```
!+
!       String List Attribute Block
!
!
!        31     24       15                0
!                +-------------------------+ +0
!                |   Pointer to first      |      SLST$V_STRING
!                |    string segment       |
!                |    or text block        |
!                +-------+-----------------+ +3
!                        |                 |
!                        | Total string    |      SLST$W_LENGTH
!                        |    length       |
!                        +-----------------+ +5   SLST$T_STRING
!
!       String List Attribute Block is made up of a Block Header + LST
!       Block + String List Attribute Block.
!-

LITERAL
    SLST$S_BLOCK_LENGTH = 5;

MACRO
    $SLST = BLOCK[SLST$S_BLOCK_LENGTH,BYTE] FIELD (SLST$Z_FIELDS)
    %;


FIELD SLST$Z_FIELDS =
    SET
        SLST$V_STRING           = [0, 0, 24, 0],
        SLST$W_LENGTH           = [3, 0, 16, 0],
        SLST$T_STRING           = [5, 0, 0, 0]
    TES;

LITERAL
    SLST$K_BASE         = SLST$S_BLOCK_LENGTH;


STRUCTURE
    SLST$ELM[I] =
        (SLST$ELM + LST$K_BASE + (I * SLST$S_BLOCK_LENGTH));
```

```
!+
!       String Segment Block
!
!        31      24       15                0
!                 +-----------------------+ +0
!                 |                       |
!                 |    Pointer to next    |      SEG$V_NEXT
!                 |    string segment     |
!                 +-----------------------+ +3   SEG$T_STRING
!
!       String Segment Block is made up of a Block Header + String
!       Segment Block.
!
!       Each string attribute points to zero or more string segment
!       blocks.  Each block contains a portion of the whole string.
!       The final string is found by catenating all the string
!       segments together.
!-

LITERAL
    SEG$S_BLOCK_LENGTH  = 3 + BLK$K_BASE;

MACRO
    $SEG = BLOCK[SEG$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
                                                 SEG$Z_FIELDS)
    %;

FIELD SEG$Z_FIELDS =
    SET
        SEG$V_NEXT              = [0+BLK$K_BASE, 0, 24, 0],
        SEG$T_STRING            = [3+BLK$K_BASE, 0, 0, 0]
    TES;

LITERAL
    SEG$K_BASE          = SEG$S_BLOCK_LENGTH;
```

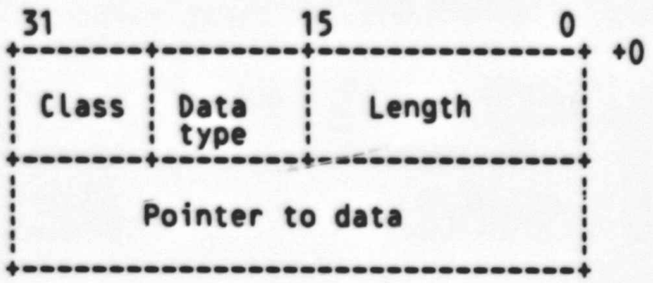
```
!+
!       Text Block
!       31                              0
!                                       +---------+ +0   TEXT$T_STRING
!                                       |         |
!                                       |  Text   |
!                                       /         \
!                                       \         /
!                                       |         |
!                                       +---------+ +n
!       Text Block is made up of a Block Header + Text BLock.
!-

LITERAL
    TEXT$S_BLOCK_LENGTH = 0 + BLK$K_BASE;

MACRO
    $TEXT = BLOCK[TEXT$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
                                                   TEXT$Z_FIELDS)
    %;

FIELD TEXT$Z_FIELDS =
    SET
        TEXT$T_STRING           = [0+BLK$K_BASE, 0, 0, 0]
    TES;

LITERAL
    TEXT$K_BASE             = TEXT$S_BLOCK_LENGTH;
```

```
!+
!       Access Control List Entry (ACL)
!
!        31              15              0
!       +---------------+---------------+ +0
!       |   Ptr to next ACL             |      ACL$V_NEXT
!       +---------------+---------------+ +3
!       |   Rights to be granted        |      ACL$L_GRANT
!       +---------------+---------------+ +7
!       |   Rights to be denied         |      ACL$L_DENY
!       +---------------+---------------+ +11
!       |   Rights to be banished       |      ACL$L_BANISH
!       +---------------+---------------+ +15
!       |  UIC group #  |  UIC member #  |      ACL$W_UIC_MEMBER
!       |               |                |      ACL$W_UIC_GROUP
!       +---------------+---------------+ +19
!       |   Ptr to first ACLC block     |      ACL$V_FIRST_ACLC
!       +-------------------------------+ +22
!
!       Each node has an Access Control List made up of zero or more
!       Access Control List Entries (ACL).
!
!       The ACL block is appended to a BLK to make an access control
!       list entry.
!-

LITERAL
    ACL$S_BLOCK_LENGTH = 22+BLK$K_BASE;

MACRO
    $ACL = BLOCK[ACL$S_BLOCK_LENGTH, BYTE] FIELD (BLK$Z_FIELDS,
                                                  ACL$Z_FIELDS)
    %;

FIELD   ACL$Z_FIELDS =
    SET
        ACL$V_NEXT              = [0+BLK$K_BASE, 0, 24, 0],
        ACL$L_GRANT             = [3+BLK$K_BASE, 0, 32, 0],
        ACL$L_DENY              = [7+BLK$K_BASE, 0, 32, 0],
        ACL$L_BANISH            = [11+BLK$K_BASE, 0, 32, 0],
        ACL$W_UIC_MEMBER        = [15+BLK$K_BASE, 0, 16, 0],
        ACL$W_UIC_GROUP         = [17+BLK$K_BASE, 0, 16, 0],
        ACL$V_FIRST_ACLC        = [19+BLK$K_BASE, 0, 24, 0]
    TES;
```

```
LITERAL
    ACL$K_BASE                  = ACL$S_BLOCK_LENGTH;
```

```
!+
!       Access Control List Criterion (ACLC)
!
!        31              15              0
!       +-------+--------+---------------+ +0
!       |       |        |               |
!       | Type  |  Ptr to next ACLC      |      ACLC$V_NEXT
!       |       |        |               |      ACLC$B_TYPE
!       +-------+--------+---------------+ +4   ACLC$T_STRING
!
!       Each ACL may have one or more Access Control List Criterion
!       (ACLC) chained from it.
!
!       Each ACLC specifies one user identification criterion for
!       the ACL entry.  The user must match all the criteria for
!       the ACL entry to apply to him.
!
!       An ACLC is appended to a BLK to form the criterion block.
!-

LITERAL
    ACLC$S_BLOCK_LENGTH = 4+BLK$K_BASE;

MACRO
    $ACLC = BLOCK[ACLC$S_BLOCK_LENGTH, BYTE] FIELD (BLK$Z_FIELDS,
                                                    ACLC$Z_FIELDS)
    %;

FIELD   ACLC$Z_FIELDS =
    SET
        ACLC$V_NEXT             = [0+BLK$K_BASE, 0, 24, 0],
        ACLC$B_TYPE             = [3+BLK$K_BASE, 0, 8, 0],
        ACLC$T_STRING           = [4+BLK$K_BASE, 0, 0, 0]
    TES;

LITERAL
    ACLC$K_BASE                 = ACLC$S_BLOCK_LENGTH;
```

String Descriptor




```
MACRO
    $DSC = BLOCK[8,BYTE]
    %;
```

```
!+
!       User Control Block (UCB)
!
!
!        31               15               0
!       +---------------------------------+ +0
!       |                                 |
!       |       Pointer to next UCB       |      UCB$A_NEXT
!       |                                 |
!       +----------------+----------------+ +4
!       |                |                |
!       |   Last User    |   First User   |      UCB$W_FIRST
!       |  Number Mapped | Number Mapped  |      UCB$W_LAST
!       +----------------+----------------+ +8   UCB$V_CCB_BASE
!       |                                 |
!       |                                 |
!       |                                 |
!       +/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/+      UCB$A_CCB
!       |                                 |
!       |                                 |
!       |                                 |
!       +---------------------------------+ +24
!
!       A user's context pointer is an origin 1 index into the CCBs in
!       the UCB list.  Each UCB can point to UCB$K_CCB number of CCBs.
!       If the user passes a context number that doesn't map to an
!       active CCB, we tell him it's an invalid context pointer.
!
!       Except in some extreme case, any image will probably never have
!       more than 4 users active at any one time. But we can handle more!
!-

LITERAL
    UCB$K_CCB_BASE              = 8,
    UCB$K_CCB                   = 4,    ! Number of CCBs/UCB
    UCB$S_BLOCK_LENGTH          = UCB$K_CCB_BASE + UCB$K_CCB * 4;

MACRO
    $UCB = BLOCK[UCB$S_BLOCK_LENGTH,BYTE] FIELD (UCB$Z_FIELDS)
    %;

FIELD   UCB$Z_FIELDS =
    SET
        UCB$A_NEXT              = [0, 0, 32, 0],
        UCB$W_FIRST             = [4, 0, 16, 0],
        UCB$W_LAST              = [6, 0, 16, 0],
        UCB$V_CCB               = [8, 0, 0, 0],
        UCB$A_CCB               = [0, 0, 32, 0]
    TES;

LITERAL
    UCB$C_FIRST = BLOCK[0, UCB$W_FIRST; UCB$S_BLOCK_LENGTH, BYTE];

STRUCTURE
    UCB$CCB[I, O, P, S, E] = (UCB$CCB+UCB$K_CCB_BASE+
                              (I-.(UCB$CCB+UCB$C_FIRST)<0,16,0>)*4+0)<P,S,E>;
```

```
!+
!       Pool Header Block (PHB)
!
!        31              15              0
!       +-------+-------+-------+-------+ +0
!       |       | Number| Low   |       |     PHB$B_POOL_TYPE
!       |       |  of   | slot  | Pool  |     PHB$B_LOW_SLOT
!       |       | slots | number| type  |     PHB$B_SLOTS
!       +-------+-------+-------+-------+ +4
!       |                               |
!       |   Ptr to last MCB in the pool |     PHB$A_LAST_MCB
!       |                               |
!       +-------------------------------+ +8
!       |                               |
!       |     Number of bytes in each   |     PHB$L_EXTENT_SIZE
!       |        secondary extent       |
!       +-------------------------------+ +12 PHB$V_FREE_BLOCK_LIST
!
!       Pools are used to provide space for various in-core block types.
!       Blocks that are related are usually allocated from the same pool.
!       This is done because pools provide good locality of reference and
!       this allocation scheme reduces page faults.
!
!       Each pool consists of one or more extents.  Each extent consists of
!       a Memory Control Block and the rest of the space allocated to the
!       extent.  Each pools has a Pool Header Block associated with it.
!       This block identifies the MCB list, as well as contains the pool's
!       free block list.
!
!       The free block list is a vector of linked lists.  Each slot in the
!       vector corresponds to a block type.  When a block is freed, it is
!       linked into its associated free list.  When a block is requested,
!       its free list is checked to see if it is non-empty.  If so, then
!       a block is allocated from it.  Otherwise, a block is allocated from
!       one of the pool's extents.
!-

LITERAL
    PHB$S_BLOCK_LENGTH          = 12;

MACRO
    $PHB = BLOCK[PHB$S_BLOCK_LENGTH, BYTE]  FIELD (PHB$Z_FIELDS)
    %;

FIELD   PHB$Z_FIELDS =
    SET
        PHB$B_POOL_TYPE         = [0, 0, 8, 0],
        PHB$B_LOW_SLOT          = [1, 0, 8, 0],
        PHB$B_SLOTS             = [2, 0, 8, 0],
        PHB$A_LAST_MCB          = [4, 0, 32, 0],
        PHB$L_EXTENT_SIZE       = [8, 0, 32, 0],
        PHB$V_FREE_BLOCK_LIST   = [12, 0, 0, 0]
    TES;
```

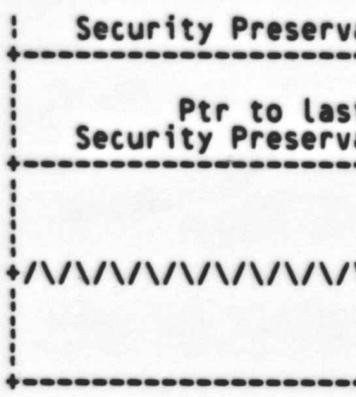
```
!+
!       Memory Control Block (MCB)
!
!        31               15               0
!       +---------------------------------+ +0
!       |                                 |
!       |  Pointer to prior memory extent |      MCB$A_PRIOR
!       |                                 |
!       +-----------------+---------------+ +4
!       |                 |               |
!       |  Remaining size |  Size of this |      MCB$W_ALLOC_SIZE
!       |  of this extent |     extent    |      MCB$W_FREE_SIZE
!       +-----------------+---------------+ +8
!
!       Each pool has one or more Memory Control Blocks associated with it.
!
!       This list serves two purposes:
!
!       1)  It points to each memory extent we asked LIB$GET_VM for.
!
!       2)  It keeps track of memory that has been allocated for
!           the pool, but never used.
!-

LITERAL
    MCB$S_BLOCK_LENGTH = 8;

MACRO
    $MCB = BLOCK[MCB$S_BLOCK_LENGTH,BYTE] FIELD (MCB$Z_FIELDS)
    %;

FIELD   MCB$Z_FIELDS =
    SET
        MCB$A_PRIOR             = [0, 0, 32, 0],
        MCB$W_ALLOC_SIZE        = [4, 0, 16, 0],
        MCB$W_FREE_SIZE         = [6, 0, 16, 0]
    TES;
```

```
!+
!       Hash Control Entry (HCE)
!
!        31               15               0
!       +---------------------------------+ +0
!       |                                 |
!       |     Forward homonym pointer     |      HCE$A_NEXT
!       |                                 |
!       +---------------------------------+ +4
!       |                                 |
!       |    Backwards homonym pointer    |      HCE$A_PRIOR
!       |                                 |
!       +-----------------+---------------+ +8
!       |                 |               |
!       | # blocks ever   | # times bucket|      HCE$W_HITS
!       |stored in bucket |   accessed    |      HCE$W_BLOCKS
!       +-----------------+---------------+ +12
!       |                 |               |
!       | Max blocks ever | # blocks now  |      HCE$W_CUR_LENGTH
!       | in bucket chain |in bucket chain|      HCE$W_MAX_LENGTH
!       +-----------------+---------------+ +16
!
!       HCEs are found in each user's CCB.  Each HCE forms one hash
!       bucket.  We keep some statistics on the buckets in an attempt
!       to tune the hash function.
!-

LITERAL
    HCE$S_BLOCK_LENGTH = 16;

MACRO
    $HCE = BLOCK[HCE$S_BLOCK_LENGTH,BYTE] FIELD (HCE$Z_FIELDS)
    %;

FIELD  HCE$Z_FIELDS =
    SET
        HCE$A_NEXT              = [0, 0, 32, 0],
        HCE$A_PRIOR             = [4, 0, 32, 0],
        HCE$W_HITS              = [8, 0, 16, 0],
        HCE$W_BLOCKS            = [10, 0, 16, 0],
        HCE$W_CUR_LENGTH        = [12, 0, 16, 0],
        HCE$W_MAX_LENGTH        = [14, 0, 16, 0]
    TES;

LITERAL
    HCE$K_HOMONYM_LIST          = BLOCK[0, HCE$A_NEXT; ,BYTE];
```

```
!+
!       Hash Control Block (HCB)
!
!        31              15              0
!       +--------------------------------+ +0
!       |     Forward homonym pointer    |      HCB$A_NEXT
!       +--------------------------------+ +4
!       |    Backwards homonym pointer   |      HCB$A_PRIOR
!       +--------------------------------+ +8
!       |           Hash key             |      HCB$V_HASH_KEY
!       +----------------+---------------+ +12
!       |       Rest of hash key         |
!       +----------------+---------------+ +16
!       |    Pointer to object block     |      HCB$A_BLOCK
!       +--------------------------------+ +20
!-

LITERAL
    HCB$K_PCB_NUMBER    = 1,    ! Hash type is page number
    HCB$K_LCCB_CODE     = 2,    ! Hash type is location code
    HCB$K_LCCB_ADDRESS  = 3,    ! Hash type is entity disk address
    HCB$K_KEY_LENGTH    = 8,    ! Size of a hash key
    HCB$S_BLOCK_LENGTH  = 20;

MACRO
    $HCB = BLOCK[HCB$S_BLOCK_LENGTH,BYTE] FIELD (HCB$Z_FIELDS)
    %;

FIELD  HCB$Z_FIELDS =
    SET
        HCB$A_NEXT              = [0, 0, 32, 0],
        HCB$A_PRIOR             = [4, 0, 32, 0],
        HCB$V_HASH_KEY          = [8, 0, 0, 0],
        HCB$A_BLOCK             = [16, 0, 32, 0]
    TES;

LITERAL
    HCB$K_HOMONYM_LIST          = BLOCK[0, HCB$A_NEXT; , BYTE];
```

Context Control Block (CCB)

31 — 15 — 0		
	Flags	CCB$W_FLAGS
Pointer to first LCB in temporary lock list		CCB$A_FIRST_LOCK
Pointer to last LCB in temporary lock list		CCB$A_LAST_LOCK
User ID	User ID for lock manager	CCB$W_LOCK_ID CCB$W_USER_ID
Next location code number to be assigned		CCB$L_NEXT_LCC
Ptr to NCB for file's CDD$TOP node		CCB$$A_CDD$TOP_NCB
NCB of CDD$LOGIN node		CCB$A_LOGIN_NCB
Ptr to current FCB for user		CCB$A_CURRENT_FCB
Ptr to highest cluster in cache		CCB$A_FIRST_CIB
Ptr to last highest cluster in cache		CCB$A_LAST_CIB
Status of current transaction		CCB$L_STATUS
Ptr to CCB's pool header		CCB$A_POOL
Ptr to first		CCB$A_FIRST_SPB



The hash table consists of a number of hash entries.
See the HCE description for the format of these entries.

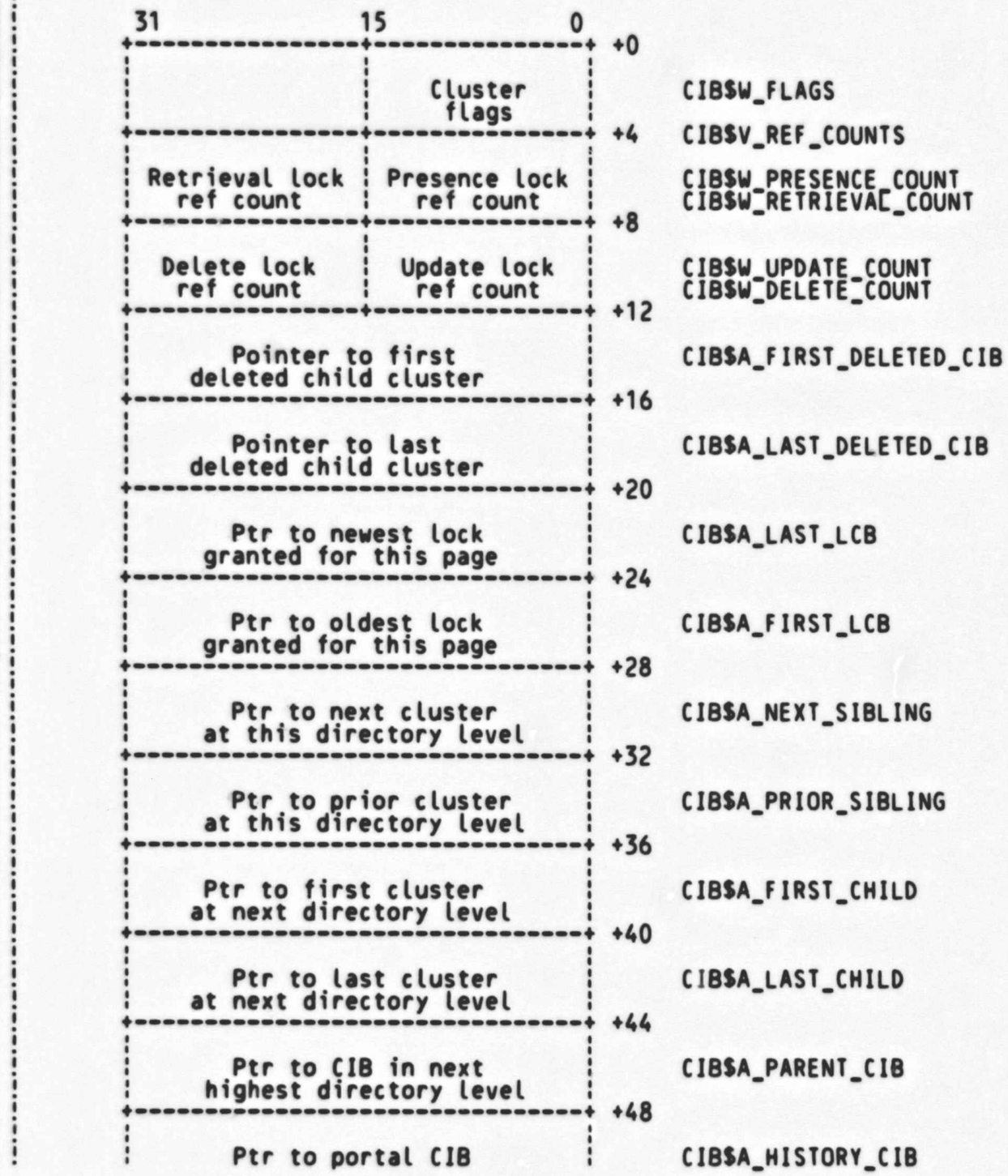
```
LITERAL
    CCB$K_HASH_TABLE_SIZE = 151,
    CCB$S_BLOCK_LENGTH    = 56 + (CCB$K_HASH_TABLE_SIZE * HCE$S_BLOCK_LENGTH);

MACRO
    $CCB = BLOCK[CCB$S_BLOCK_LENGTH,BYTE] FIELD (CCB$Z_FIELDS)
    %;

FIELD  CCB$Z_FIELDS =
    SET
        CCB$W_FLAGS             = [0, 0, 16, 0],
        CCB$V_CORRUPT           = [0, 0, 1, 0],         ! Stream is corrupt
        CCB$A_FIRST_LOCK        = [4, 0, 32, 0],
        CCB$A_LAST_LOCK         = [8, 0, 32, 0],
        CCB$W_LOCK_ID           = [12, 0, 16, 0],
        CCB$W_USER_ID           = [14, 0, 16, 0],
        CCB$L_NEXT_LCC          = [16, 0, 32, 0],
        CCB$A_CDD$TOP_NCB       = [20, 0, 32, 0],
        CCB$A_LOGIN_NCB         = [24, 0, 32, 0],
        CCB$A_CURRENT_FCB       = [28, 0, 32, 0],
        CCB$A_FIRST_CIB         = [32, 0, 32, 0],
        CCB$A_LAST_CIB          = [36, 0, 32, 0],
        CCB$L_STATUS            = [40, 0, 32, 0],
        CCB$A_POOL              = [44, 0, 32, 0],
        CCB$A_FIRST_SPB         = [48, 0, 32, 0],
        CCB$A_LAST_SPB          = [52, 0, 32, 0],
        CCB$V_HASH_BASE         = [56, 0, 0, 0]
    TES;

LITERAL
    CCB$K_LOCK_LIST     = BLOCK[0, CCB$A_FIRST_LOCK; , BYTE],
    CCB$K_CLUSTER_LIST  = BLOCK[0, CCB$A_FIRST_CIB; , BYTE],
    CCB$S_HASH_BASE     = BLOCK[0, CCB$V_HASH_BASE; , BYTE],
    CCB$K_SPB_LIST      = BLOCK[0, CCB$A_FIRST_SPB; , BYTE];

STRUCTURE
    CCB$HASH[I, O, P, S, E] =
```

```
        (CCB$HASH+CCB$S_HASH_BASE+0+(I*HCE$S_BLOCK_LENGTH))<P,S,E>;

LITERAL
    CCB$M_CORRUPT       = 1^1 - 1^0;    ! Stream is corrupt
```

```
!+
!       Page Control Block (PCB)
!
!
!        31              15              0
!       +--------------------------------+ +0
!       |          Page Number           |      PCB$L_PAGE_NUMBER
!       +----------------+---------------+ +4
!       | Paging file    |  Page flags   |      PCB$W_FLAGS
!       | block number   |               |      PCB$W_BLOCK_NUMBER
!       +----------------+---------------+ +8
!       |    Pointer to page             |      PCB$A_BUFFER
!       |  when in virtual memory        |
!       +--------------------------------+ +12
!       |    Pointer to PCB of           |      PCB$A_NEXT_CLUSTER
!       |  next page in cluster          |
!       +--------------------------------+ +16
!       |    Pointer to CIB of           |      PCB$A_PORTAL_CIB
!       |  cluster's portal page         |
!       +--------------------------------+ +20
!       |    Ptr to page's FCB           |      PCB$A_FCB
!       +--------------------------------+ +24
!       |    Pointer to page's BCB       |      PCB$A_BCB
!       +--------------------------------+ +28
!       |    Pointer to first            |      PCB$A_FIRST_LCCB
!       |  LCCB in page cluster          |
!       +--------------------------------+ +32
!       |    Pointer to last             |      PCB$A_LAST_LCCB
!       |  LCCB in page cluster          |
!       +--------------------------------+ +36
!
!       A Page Control Block (PCB) exists for every page in the cache,
!       and for pages that only have presence locks on them (thus they're
!       not in the staging cache).  Portal pages have a Cluster Information
!       Block (CIB) appending to their PCB.
!-

LITERAL
    PCB$S_BLOCK_LENGTH = 36;

MACRO
    $PCB = BLOCK[PCB$S_BLOCK_LENGTH,BYTE] FIELD (PCB$Z_FIELDS)
    %;
```

```
FIELD   PCB$Z_FIELDS =
    SET
        PCB$L_PAGE_NUMBER       = [0, 0, 32, 0],
        PCB$W_FLAGS             = [4, 0, 16, 0],
        PCB$V_MODIFIED          = [4, 0, 1, 0],     ! Must write back to dict.
        PCB$V_NEW_PAGE          = [4, 3, 1, 0],     ! Page was in free chain
        PCB$V_READ_ONLY         = [4, 4, 1, 0],     ! Page is read only
        PCB$V_FREE_PAGE         = [4, 5, 1, 0],     ! Page is a free page
        PCB$V_AVAILABLE         = [4, 6, 1, 0],     ! Page available in cache
        PCB$V_PORTAL_PAGE       = [4, 7, 1, 0],     ! Page is portal page, CIB follows
        PCB$W_BLOCK_NUMBER      = [6, 0, 16, 0],
        PCB$A_BUFFER            = [8, 0, 32, 0],
        PCB$A_NEXT_CLUSTER      = [12, 0, 32, 0],
        PCB$A_PORTAL_CIB        = [16, 0, 32, 0],
        PCB$A_FCB               = [20, 0, 32, 0],
        PCB$A_BCB               = [24, 0, 32, 0],
        PCB$A_FIRST_LCCB        = [28, 0, 32, 0],
        PCB$A_LAST_LCCB         = [32, 0, 32, 0]
    TES;

LITERAL
    PCB$K_LCCB_LIST             = BLOCK[0, PCB$A_FIRST_LCCB; , BYTE];

LITERAL
    PCB$M_MODIFIED              = 1^1 - 1^0,    ! Must write back to dict.
    PCB$M_NEW_PAGE              = 1^4 - 1^3,    ! Page was in free chain
    PCB$M_READ_ONLY             = 1^5 - 1^4,    ! Page is read only
    PCB$M_FREE_PAGE             = 1^6 - 1^5,    ! Page is a free page
    PCB$M_AVAILABLE             = 1^7 - 1^6,    ! Page available in cache
    PCB$M_PORTAL_PAGE           = 1^8 - 1^7;    ! Portal page, CIB follows
```

Cluster Information Block (CIB)



```
!       !in cluster holding history list !
!       +--------------------------------+ +52
!       !                                !
!       !       Ptr to NCB for the       !      CIB$A_OWNER_NCB
!       !  node that owns this cluster   !
!       +--------------------------------+ +56
!       !                                !
!       !  Ptr to cluster's pool header  !      CIB$A_POOL
!       !                                !
!       +--------------------------------+ +60
!
!       Each clusters' portal page has a CIB associated with it.
!       This block is physically appended to the PCB of the cluster's
!       portal page.
!
!       The CIB also points to the cluster's pool header.
!
!       The CIB$V_REF_COUNTS lists must have each of the lock ref counts
!       in the same order as the LOCK$K_xxx lock request symbols.
!-

LITERAL
    CIB$S_BLOCK_LENGTH = 60 + PCB$S_BLOCK_LENGTH;

MACRO
    $CIB = BLOCK[CIB$S_BLOCK_LENGTH,BYTE] FIELD (CIB$Z_FIELDS,
                                                 PCB$Z_FIELDS)
    %;

FIELD   CIB$Z_FIELDS =
    SET
        CIB$W_FLAGS             = [0+PCB$S_BLOCK_LENGTH, 0, 16, 0],
        CIB$V_LOCKED_DELETE     = [0+PCB$S_BLOCK_LENGTH, 0, 1, 0],
        CIB$V_LOCKED_UPDATE     = [0+PCB$S_BLOCK_LENGTH, 1, 1, 0],
        CIB$V_LOCKED_RETRIEVAL  = [0+PCB$S_BLOCK_LENGTH, 2, 1, 0],
        CIB$V_LOCKED_PRESENCE   = [0+PCB$S_BLOCK_LENGTH, 3, 1, 0],
        CIB$V_LOCKED            = [0+PCB$S_BLOCK_LENGTH, 0, 4, 0],
        CIB$V_LOGIN             = [0+PCB$S_BLOCK_LENGTH, 4, 1, 0],
        CIB$V_COMPLETE          = [0+PCB$S_BLOCK_LENGTH, 5, 1, 0],
        CIB$V_NEW               = [0+PCB$S_BLOCK_LENGTH, 6, 1, 0],
        CIB$V_HISTORY           = [0+PCB$S_BLOCK_LENGTH, 7, 1, 0],
        CIB$V_REF_COUNTS        = [4+PCB$S_BLOCK_LENGTH, 0, 0, 0],
        CIB$W_PRESENCE_COUNT    = [4+PCB$S_BLOCK_LENGTH, 0, 16, 0],
        CIB$W_RETRIEVAL_COUNT   = [6+PCB$S_BLOCK_LENGTH, 0, 16, 0],
        CIB$W_UPDATE_COUNT      = [8+PCB$S_BLOCK_LENGTH, 0, 16, 0],
        CIB$W_DELETE_COUNT      = [10+PCB$S_BLOCK_LENGTH, 0, 16, 0],
        CIB$A_FIRST_DELETED_CIB = [12+PCB$S_BLOCK_LENGTH, 0, 32, 0],
        CIB$A_LAST_DELETED_CIB  = [16+PCB$S_BLOCK_LENGTH, 0, 32, 0],
        CIB$A_LAST_LCB          = [20+PCB$S_BLOCK_LENGTH, 0, 32, 0],
        CIB$A_FIRST_LCB         = [24+PCB$S_BLOCK_LENGTH, 0, 32, 0],
        CIB$A_NEXT_SIBLING      = [28+PCB$S_BLOCK_LENGTH, 0, 32, 0],
        CIB$A_PRIOR_SIBLING     = [32+PCB$S_BLOCK_LENGTH, 0, 32, 0],
        CIB$A_FIRST_CHILD       = [36+PCB$S_BLOCK_LENGTH, 0, 32, 0],
        CIB$A_LAST_CHILD        = [40+PCB$S_BLOCK_LENGTH, 0, 32, 0],
        CIB$A_PARENT_CIB        = [44+PCB$S_BLOCK_LENGTH, 0, 32, 0],
```

```
        CIB$A_HISTORY_CIB       = [48+PCB$S_BLOCK_LENGTH, 0, 32, 0],
        CIB$A_OWNER_NCB         = [52+PCB$S_BLOCK_LENGTH, 0, 32, 0],
        CIB$A_POOL              = [56+PCB$S_BLOCK_LENGTH, 0, 32, 0]
    TES;

LITERAL
    CIB$K_LOCK_LIST             = BLOCK[0, CIB$A_LAST_LCB; , BYTE],
    CIB$K_SIBLING_LIST          = BLOCK[0, CIB$A_NEXT_SIBLING; , BYTE],
    CIB$K_CHILD_LIST            = BLOCK[0, CIB$A_FIRST_CHILD; , BYTE];

LITERAL
    CIB$M_LOCKED_DELETE         = 1^1 - 1^0,
    CIB$M_LOCKED_UPDATE         = 1^2 - 1^1,
    CIB$M_LOCKED_RETRIEVAL      = 1^3 - 1^2,
    CIB$M_LOCKED_PRESENCE       = 1^4 - 1^3,
    CIB$M_LOCKED                = 1^4 - 1^0,
    CIB$M_LOGIN                 = 1^5 - 1^4,    ! Cluster is on login path
    CIB$M_COMPLETE              = 1^6 - 1^5,    ! Cluster is complete in cache
    CIB$M_NEW                   = 1^7 - 1^6,    ! Cluster read in this transaction
    CIB$M_HISTORY               = 1^8 - 1^7;    ! History list is in cluster
```

```
!+
!       Lock Control Block (LCB)
!
!        31              15              0
!       +----------------+----------------+ +0
!       |  Block's flags |   Lock Type    |      LCB$W_LOCK_TYPE
!       |                |                |      LCB$W_FLAGS
!       +----------------+----------------+ +4
!       |      Ptr to prior LCB           |      LCB$A_PRIOR_LCB
!       +---------------------------------+ +8
!       |      Ptr to next LCB            |      LCB$A_NEXT_LCB
!       +---------------------------------+ +12
!       |      Ptr to locked CIB          |      LCB$A_CIB
!       +---------------------------------+ +16
!       |      Ptr to next lock           |      LCB$A_NEXT_LOCK
!       |   in transaction lock list      |
!       +---------------------------------+ +20
!       |      Ptr to prior lock          |      LCB$A_PRIOR_LOCK
!       |   in transaction lock list      |
!       +----------------+----------------+ +24
!
!       LCBs are used to keep track of which locks exist on a cluster (CIB).
!
!       Each LCB is linked to its CIB, and to other LCBs for that cluster.
!       When an LCB is granted, it is placed in the transaction's
!       lock list until the transaction terminates.
!-

LITERAL
    LCB$S_BLOCK_LENGTH = 24;

MACRO
    $LCB = BLOCK[LCB$S_BLOCK_LENGTH,BYTE] FIELD (LCB$Z_FIELDS)
    %;

FIELD   LCB$Z_FIELDS =
    SET
        LCB$W_LOCK_TYPE         = [0, 0, 16, 0],
        LCB$W_FLAGS             = [2, 0, 16, 0],
        LCB$V_CURRENT           = [2, 1, 1, 0], ! Allocated in current transaction
        LCB$A_PRIOR_LCB         = [4, 0, 32, 0],
        LCB$A_NEXT_LCB          = [8, 0, 32, 0],
        LCB$A_CIB               = [12, 0, 32, 0],
        LCB$A_NEXT_LOCK         = [16, 0, 32, 0],
        LCB$A_PRIOR_LOCK        = [20, 0, 32, 0]
    TES;
```

```
LITERAL
    LCB$K_LOCK_LIST             = BLOCK[0, LCB$A_PRIOR_LCB; , BYTE],
    LCB$K_TEMP_LOCK_LIST        = BLOCK[0, LCB$A_NEXT_LOCK; , BYTE];

LITERAL
    LCB$M_CURRENT               = 1^2 - 1^1;    ! Allocated in current transaction
```

```
!+
!       Location Code Control Block (LCCB)
!
!        31              15              0
!       +--------------------------------+ +0
!       |      Location code number      |      LCCB$L_LOCATION_CODE
!       +-------+--------+---------------+ +4
!       |       |        |               |      LCCB$W_FLAGS
!       | LCCB  | Block  |  LCCB flags   |      LCCB$B_TYPE
!       | type  |  type  |               |      LCCB$B_LCCB_TYPE
!       +-------+--------+---------------+ +8
!       |      Object's disk address     |      LCCB$L_OBJECT/LCCB$V_OBJECT
!       +--------------------------------+ +12
!       |       Pointer to object        |
!       |       in virtual memory        |      LCCB$A_OBJECT
!       +--------------------------------+ +16
!       |        Pointer to PCB          |
!       |  for page which holds object   |      LCCB$A_PCB
!       +--------------------------------+ +20
!       |       Pointer to next          |
!       |       LCCB in cluster          |      LCCB$A_NEXT_LCCB       ! Virt addr
!       +--------------------------------+ +24                        ! must be
!       |       Pointer to prior         |                           ! 0 mod 4,
!       |       LCCB in cluster          |      LCCB$A_PRIOR_LCCB      ! for queue
!       +---------+----------------------+ +28                        ! instructions
!                 |   DPTR of last block |
!                 | returned by FIND_NEXT|      LCCB$V_LAST_BLOCK
!                 |  (0 if not used yet) |
!                 +----------------------+ +31
!
!       Location codes are used to provide a convenient means for the
!       user to identify a particular object.  Each location code is
!       associated with an LCCB.  The following objects may be
!       assigned location code:
!
!       1)  Entities
!
!       2)  Lists
!
!       3)  Nodes
!-

LITERAL
    LCCB$S_BLOCK_LENGTH = 31;

MACRO
```

```
    $LCCB = BLOCK[LCCB$S_BLOCK_LENGTH,BYTE] FIELD (LCCB$Z_FIELDS)
    %;

FIELD   LCCB$Z_FIELDS =
    SET
        LCCB$L_LOCATION_CODE    = [0, 0, 32, 0],
        LCCB$W_FLAGS            = [4, 0, 16, 0],
        LCCB$V_AVAILABLE        = [4, 0, 1, 0],
        LCCB$V_MUST_SCAN        = [4, 1, 1, 0],
        LCCB$V_GHOST            = [4, 2, 1, 0],
        LCCB$V_DIRECTORY        = [4, 3, 1, 0],
        LCCB$V_TERMINAL         = [4, 4, 1, 0],
        LCCB$V_ENTITY_ATT       = [4, 5, 1, 0],
        LCCB$V_ENTITY_LIST_ATT  = [4, 6, 1, 0],
        LCCB$V_ENTITY_LIST      = [4, 7, 1, 0],
        LCCB$V_STRING_LIST_ATT  = [4, 8, 1, 0],
        LCCB$B_TYPE             = [6, 0, 8, 0],
        LCCB$B_LCCB_TYPE        = [7, 0, 8, 0],
        LCCB$L_OBJECT           = [8, 0, 32, 0],
        LCCB$V_OBJECT           = [8, 0, 24, 0],
        LCCB$A_OBJECT           = [12, 0, 32, 0],
        LCCB$A_PCB              = [16, 0, 32, 0],
        LCCB$A_NEXT_LCCB        = [20, 0, 32, 0],
        LCCB$A_PRIOR_LCCB       = [24, 0, 32, 0],
        LCCB$V_LAST_BLOCK       = [28, 0, 24, 0]
    TES;

LITERAL
    LCCB$K_LCCB_LIST            = BLOCK[0, LCCB$A_NEXT_LCCB; , BYTE];

LITERAL
    LCCB$M_AVAILABLE            = 1^1 - 1^0,    ! Object's virtual addr is known
    LCCB$M_MUST_SCAN            = 1^2 - 1^1,    ! Protection tree must be scanned
    LCCB$M_GHOST                = 1^3 - 1^2,    ! LCCB may not be fetched by LCC
    LCCB$M_DIRECTORY            = 1^4 - 1^3,    ! Directory NCB
    LCCB$M_TERMINAL             = 1^5 - 1^4,    ! Terminal NCB
    LCCB$M_ENTITY_ATT           = 1^6 - 1^5,    ! Entity attribute LCCB
    LCCB$M_ENTITY_LIST_ATT      = 1^7 - 1^6,    ! Entity list attribute LCCB
    LCCB$M_ENTITY_LIST          = 1^8 - 1^7,    ! Entity list ECCB
    LCCB$M_STRING_LIST_ATT      = 1^9 - 1^8,    ! String list LCCB

    LCCB$K_LCCB_TYPE_FIRST      = 1,
    LCCB$K_NCB                  = 1,            ! LCCB includes NCB
    LCCB$K_ECCB                 = 2,            ! LCCB includes ECCB
    LCCB$K_LCCB                 = 3,            ! LCCB
    LCCB$K_LCCB_TYPE_LAST       = 3;
```

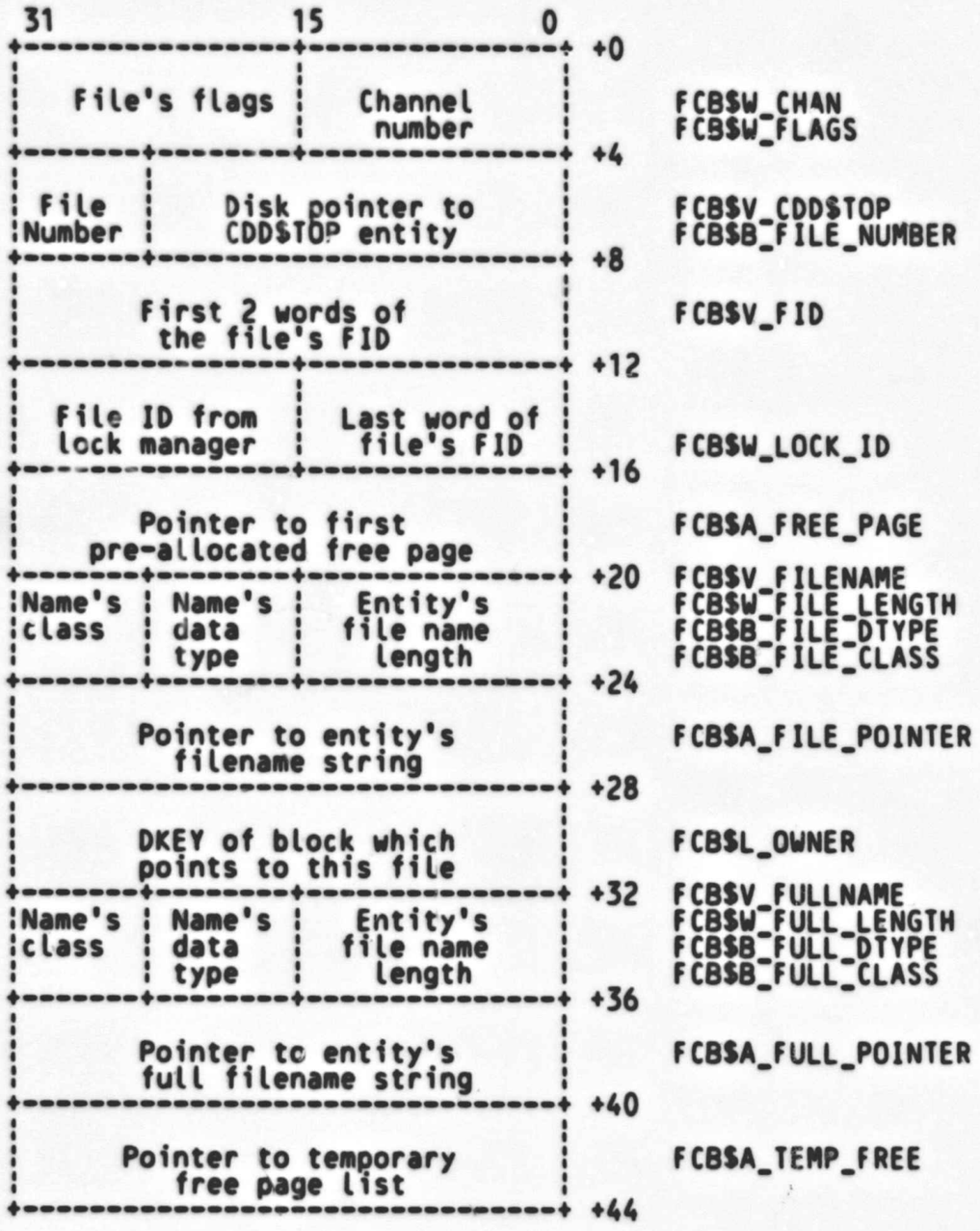
```
!+
!       Entity Cell Control Block (ECCB)
!
!
!        31              15              0
!                        +---------------+ +0
!                        |               |
!                        |  Cell number  |      ECCB$W_CELL
!                        |               |
!                +-------+---------------+ +2
!                |       |               |
!                |  DPTR of this cell's  |      ECCB$V_LST
!                |     LST block         |
!        +-------+-------+               | +5
!        |               |               |
!        |    Ptr to LCCB of the         |      ECCB$A_PARENT_LCCB
!        |  list that owns this cell     |
!        +-------------------------------+ +9
!
!       Each cell in an entity list may be assigned a location code.
!
!       This block is appended to the location code's LCCB to name the
!       specific cell represented by the location code.
!-

LITERAL
    ECCB$S_BLOCK_LENGTH =       LCCB$S_BLOCK_LENGTH + 9;

MACRO
    $ECCB = BLOCK[ECCB$S_BLOCK_LENGTH,BYTE] FIELD (ECCB$Z_FIELDS,
                                                   LCCB$Z_FIELDS)
    %;

FIELD   ECCB$Z_FIELDS =
    SET
        ECCB$W_CELL             = [0+LCCB$S_BLOCK_LENGTH, 0, 16, 0],
        ECCB$V_LST              = [2+LCCB$S_BLOCK_LENGTH, 0, 24, 0],
        ECCB$A_PARENT_LCCB      = [5+LCCB$S_BLOCK_LENGTH, 0, 32, 0]
    TES;
```

```
!+
!       Node Control Block (NCB)
!
!        31              15              0
!       +----------------+----------------+ +0
!       |                                 |      NCB$L_ACCESS_RIGHTS
!       |      User's access rights       |
!       |         to this node            |
!       +---------------------------------+ +4
!       |                                 |      NCB$L_ACCESS_GRANTED
!       |      Access rights granted      |
!       |       by this node's ACL        |
!       +---------------------------------+ +8
!       |                                 |      NCB$L_ACCESS_DENIED
!       |      Access rights denied       |
!       |       by this node's ACL        |
!       +---------------------------------+ +12
!       |                                 |      NCB$L_ACCESS_BANISHED
!       |     Access rights banished      |
!       |       by this node's ACL        |
!       +---------------------------------+ +16
!       |                                 |      NCB$A_PARENT_NCB
!       |       Pointer to parent         |
!       |          NCB block              |
!       +--------+--------+---------------+ +20  NCB$V_PASSWORD
!       |Pswrd's | Pswrd's|    Entity's   |      NCB$W_PSW_LENGTH
!       |class   | data   |   password    |      NCB$B_PSW_DTYPE
!       |        | type   |    length     |      NCB$B_PSW_CLASS
!       +--------+--------+---------------+ +24
!       |                                 |      NCB$A_PSW_POINTER
!       |       Pointer to entity's       |
!       |         password string         |
!       +---------------------------------+ +28  NCB$V_NAME
!       |Name's  | Name's |               |      NCB$W_NAME_LENGTH
!       |class   | data   | Entity name's |      NCB$B_NAME_DTYPE
!       |        | type   |    length     |      NCB$B_NAME_CLASS
!       +--------+--------+---------------+ +32
!       |                                 |      NCB$A_NAME_POINTER
!       |       Pointer to entity's       |
!       |         name string             |
!       +---------------------------------+ +36
!       |                                 |      NCB$A_SPB
!       |            Pointer to           |
!       |   Security Preservation Block   |
!       +---------------------------------+ +40
!
!       The LCCB for a node is followed by an NCB.  This gives additional
!       information needed for the node.
!-

LITERAL
    NCB$S_BLOCK_LENGTH = 40+LCCB$S_BLOCK_LENGTH;

MACRO
```

```
    $NCB = BLOCK[NCB$S_BLOCK_LENGTH,BYTE] FIELD (LCCB$Z_FIELDS,
                                                 NCB$Z_FIELDS)
    %;

FIELD   NCB$Z_FIELDS =
    SET
        NCB$L_ACCESS_RIGHTS     = [0+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
        NCB$L_ACCESS_GRANTED    = [4+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
        NCB$L_ACCESS_DENIED     = [8+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
        NCB$L_ACCESS_BANISHED   = [12+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
        NCB$A_PARENT_NCB        = [16+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
        NCB$V_PASSWORD          = [20+LCCB$S_BLOCK_LENGTH, 0, 0, 0],
        NCB$W_PSW_LENGTH        = [20+LCCB$S_BLOCK_LENGTH, 0, 16, 0],
        NCB$B_PSW_DTYPE         = [22+LCCB$S_BLOCK_LENGTH, 0, 8, 0],
        NCB$B_PSW_CLASS         = [23+LCCB$S_BLOCK_LENGTH, 0, 8, 0],
        NCB$A_PSW_POINTER       = [24+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
        NCB$V_NAME              = [28+LCCB$S_BLOCK_LENGTH, 0, 0, 0],
        NCB$W_NAME_LENGTH       = [28+LCCB$S_BLOCK_LENGTH, 0, 16, 0],
        NCB$B_NAME_DTYPE        = [30+LCCB$S_BLOCK_LENGTH, 0, 8, 0],
        NCB$B_NAME_CLASS        = [31+LCCB$S_BLOCK_LENGTH, 0, 8, 0],
        NCB$A_NAME_POINTER      = [32+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
        NCB$A_SPB               = [36+LCCB$S_BLOCK_LENGTH, 0, 32, 0]
    TES;
```

!+

File Control Block (FCB)



Each active (open) dictionary file has an FCB.

The FCB$L_OWNER field holds the DKEY of the attribute block (FIL) which pointed to the file. The primary dictionary file has key 0, while files that are not currently pointed to have a value of -1

```
!       in the owner field.
!+

LITERAL
    FCB$K_FILE_LIMIT    = 255,
    FCB$S_BLOCK_LENGTH = 44;

MACRO
    $FCB = BLOCK[FCB$S_BLOCK_LENGTH,BYTE] FIELD (FCB$Z_FIELDS)
    %;

FIELD    FCB$Z_FIELDS =
    SET
        FCB$W_CHAN              = [0, 0, 16, 0],
        FCB$W_FLAGS             = [2, 0, 16, 0],
        FCB$V_READ_ONLY         = [2, 0, 1, 0],
        FCB$V_ROOT              = [2, 1, 1, 0],
        FCB$V_CDD$TOP           = [4, 0, 24, 0],
        FCB$B_FILE_NUMBER       = [7, 0, 8, 0],
        FCB$V_FID               = [8, 0, 0, 0],
        FCB$W_LOCK_ID           = [14, 0, 16, 0],
        FCB$A_FREE_PAGE         = [16, 0, 32, 0],
        FCB$V_FILENAME          = [20, 0, 0, 0],
        FCB$W_FILE_LENGTH       = [20, 0, 16, 0],
        FCB$B_FILE_DTYPE        = [22, 0, 8, 0],
        FCB$B_FILE_CLASS        = [23, 0, 8, 0],
        FCB$A_FILE_POINTER      = [24, 0, 32, 0],
        FCB$L_OWNER             = [28, 0, 32, 0],
        FCB$V_FULLNAME          = [32, 0, 0, 0],
        FCB$W_FULL_LENGTH       = [32, 0, 16, 0],
        FCB$B_FULL_DTYPE        = [34, 0, 8, 0],
        FCB$B_FULL_CLASS        = [35, 0, 8, 0],
        FCB$A_FULL_POINTER      = [36, 0, 32, 0],
        FCB$A_TEMP_FREE         = [40, 0, 32, 0]
    TES;

LITERAL
    FCB$M_READ_ONLY     = 1^1 - 1^0,    ! File can only be opened for read
    FCB$M_ROOT          = 1^2 - 1^1;    ! FCB is the root dictionary file
```

```
!+
!       Pre-Allocated Page Block (PAPB)
!
!
!        31               15               0
!       +---------------------------------+ +0
!       |                                 |
!       |       Pointer to next           |      PAPB$A_NEXT
!       |            BCB                  |
!       +---------------------------------+ +4
!       |                                 |
!       |       Number of free page       |      PAPB$L_NUMBER
!       |                                 |
!       +---------------------------------+ +8
!
!       Pre-allocated pages are represented by PAPBs linked onto the
!       FCB in which the pages reside.  Each PAPB has the page number
!       of the free page it represents.
!-

LITERAL
    PAPB$S_BLOCK_LENGTH = 8;

MACRO
    $PAPB = BLOCK[PAPB$S_BLOCK_LENGTH,BYTE] FIELD (PAPB$Z_FIELDS)
    %;

FIELD   PAPB$Z_FIELDS =
    SET
        PAPB$A_NEXT                     = [0, 0, 32, 0],
        PAPB$L_NUMBER                   = [4, 0, 32, 0]
    TES;
```

```
!+
!       Buffer Control Block (BCB)
!
!        31              15              0
!       +--------------------------------+ +0
!       |                                |
!       |       Pointer to next          |      BCB$A_NEXT
!       |            BCB                 |
!       +--------------------------------+ +4
!       |                                |
!       |       Pointer to prior         |      BCB$A_PRIOR
!       |            BCB                 |
!       +--------------------------------+ +8
!       |                                |
!       |       Pointer to buffer        |      BCB$A_BUFFER
!       |                                |
!       +--------------------------------+ +12
!       |                                |
!       |       Pointer to PCB           |      BCB$A_PCB
!       |    for page within buffer      |
!       +--------------------------------+ +16
!
!       The buffer control blocks (BCBs) are used to control the pages
!       that are in memory.
!
!       The BCBs are linked into a queue.  When a buffer is needed, the
!       last buffer in the queue is assigned.
!
!       The associated PCB contains information about the page and the
!       buffer.  If a page is modified while in the buffer, the page
!       is written back to the work file before the buffer is reused.
!-

LITERAL
    BCB$K_NUMBER        = 16,   ! Number of in-core buffers
    BCB$S_BLOCK_LENGTH  = 16;

MACRO
    $BCB = BLOCK[BCB$S_BLOCK_LENGTH,BYTE] FIELD (BCB$Z_FIELDS)
    %;

FIELD  BCB$Z_FIELDS =
    SET
        BCB$A_NEXT              = [0, 0, 32, 0],
        BCB$A_PRIOR             = [4, 0, 32, 0],
        BCB$A_BUFFER            = [8, 0, 32, 0],
        BCB$A_PCB               = [12, 0, 32, 0]
    TES;

LITERAL
    BCB$K_BUFFER_LIST           = BLOCK[0, BCB$A_NEXT; , BYTE];
```

```
!+
!       User Retrieval List Entry Block
!       31        23        15        7         0
!       +---------+---------+---------+---------+ +0
!       |                   |                   |   URLC$B_COUNT
!       |      Unused       |       Count       |
!       |                   |                   |   URLC$V_UNUSED
!       +---------+---------+---------+---------+ +4
!
!
!       User Retrieval List Entry Block
!       31        23        15        7         0
!       +---------+---------+---------+---------+ +0
!       |                                       |
!       |          Attribute Name               |   URLE$L_ATT/URLE$L_CELL
!       |                or                     |
!       |           Cell Number                 |
!       +---------+---------+---------+---------+ +4
!       |                   |                   |   URLE$W_DATA_TYPE
!       |     Reserved      |       Data        |
!       |                   |       Type        |   URLE$W_RESERVE
!       |                   |     Expected      |
!       +---------+---------+---------+---------+ +8
!       |                                       |
!       |          Returned Status              |   URLE$L_STATUS
!       |                                       |
!       +---------+---------+---------+---------+ +12
!       |                   |                   |   URLE$W_USER_LENGTH
!       |  Returned Length  |      Length       |
!       |                   |        of         |   URLE$W_RETURN_LENGTH
!       |                   |    User Buffer    |
!       +---------+---------+---------+---------+ +16
!       |                                       |
!       |          Address of                   |   URLE$A_USER_BUFFER
!       |       User Buffer or                  |
!       |      String Descriptor                |
!       +---------+---------+---------+---------+ +20

LITERAL
    URLC$S_BLOCK_LENGTH = 4,
    URLE$S_BLOCK_LENGTH = 20;

MACRO
    $URLC = BLOCK[URLC$S_BLOCK_LENGTH,BYTE] FIELD (URLC$Z_FIELDS)
    %;

FIELD   URLC$Z_FIELDS =
    SET
        URLC$B_COUNT    = [0, 0, 8, 0],
        URLC$V_UNUSED   = [1, 0, 24, 0]
    TES;


MACRO
    $URLE = BLOCK[URLE$S_BLOCK_LENGTH,BYTE] FIELD (URLE$Z_FIELDS)
    %;

FIELD   URLE$Z_FIELDS =
    SET
```

```
        URLE$L_ATT              = [0, 0, 32, 0],
        URLE$L_CELL             = [0, 0, 32, 0],
        URLE$W_DATA_TYPE        = [4, 0, 16, 0],
        URLE$W_RESERVE          = [6, 0, 16, 0],
        URLE$L_STATUS           = [8, 0, 32, 0],
        URLE$W_USER_LENGTH      = [12, 0, 16, 0],
        URLE$W_RETURN_LENGTH    = [14, 0, 16, 0],
        URLE$A_USER_BUFFER      = [16, 0, 32, 0]
    TES;

STRUCTURE
    URL$BLOCK [I] = (URL$BLOCK + URLC$S_BLOCK_LENGTH +
                        (I * URLE$S_BLOCK_LENGTH));
```

```
!+
!       Write Control Block (WCB)
!
!        31                15                0
!       +-----------------+-----------------+ +0
!       |      Flags      |Number of pages  |      WCB$W_SIZE
!       |                 |in write clstr   |      WCB$W_FLAGS
!       +-----------------+-----------------+ +4
!       |        Ptr to page's              |      WCB$A_OLD_BUF
!       |       permanant buffer            |
!       +-----------------------------------+ +8
!       |        Ptr to page's              |      WCB$A_NEW_BUF
!       |    location in write buffer       |
!       +-----------------+-----------------+ +12  WCB$V_IO_STATUS
!       |  2nd word of    |  1st word of    |      WCB$W_IO_STATUS
!       |  I/O status blk |  I/O status blk |
!       +-----------------+-----------------+ +16
!       |  4th word of    |  3rd word of    |
!       |  I/O status blk |  I/O status blk |
!       +-----------------+-----------------+ +20
!
!-

LITERAL
    WCB$S_BLOCK_LENGTH = 20;

MACRO
    $WCB = BLOCK[WCB$S_BLOCK_LENGTH, BYTE] FIELD (WCB$Z_FIELDS)
    %;

FIELD   WCB$Z_FIELDS =
    SET
        WCB$W_SIZE              = [0, 0, 16, 0],
        WCB$W_FLAGS             = [2, 0, 16, 0],
        WCB$V_WRITABLE          = [2, 0, 1, 0], ! Write group to file
        WCB$V_GROUP             = [2, 1, 1, 0], ! Group of pages
        WCB$A_OLD_BUF           = [4, 0, 32, 0],
        WCB$A_NEW_BUF           = [8, 0, 32, 0],
        WCB$V_IO_STATUS         = [12, 0, 0, 0],
        WCB$W_IO_STATUS         = [12, 0, 16, 0]
    TES;

LITERAL
    WCB$M_WRITABLE      = 1^1 - 1^0,    ! Write group to file
    WCB$M_GROUP         = 1^2 - 1^1;    ! Group of pages
```

```
!+
!       Attribute Name Block (ATNM)
!
!    31 30 29 28 27 26                  16 15                                0
!   +--+--+--+--+--+--+--------------------+----------------------------------+
!   |  |  |  |  |  |  |                    |                                  |
!   |  |  |  |  |  |  |   Protocol number  |        Attribute number          |
!   |  |  |  |  |  |  |                    |                                  |
!   +--+--+--+--+--+--+--------------------+----------------------------------+
!      |  |  |  |  |
!      |  |  |  |  +---  CDD-defined attribute
!      |  |  |  +------  Global attribute
!      |  |  +---------  Customer-defined attribute
!      |  +------------  Reserved
!      +---------------  Reserved
!
!       The attribute name block defines the break down of the
!       attribute name.  Bits 0 to 15 contains the number.  Bits 16
!       to 26 contains the protocol.  The remaining bits are flags
!       define the type of attribute.  Bit 27 on implies a system
!       defined attribute.  Bit 28 on implies a global defined attribute.
!       Bit 29 on implies a customer defined attribute.
!
!       NOTE: If the high order word of the attribute name block is 0
!             the block is describing a cell. That is the protocol will
!             equal 0 and the flag bits will equal 0.
!-

LITERAL
    ATNM$S_BLOCK_LENGTH = 4;

MACRO
    $ATNM = BLOCK[ATNM$S_BLOCK_LENGTH,BYTE] FIELD (ATNM$Z_FIELDS)
    %;

FIELD   ATNM$Z_FIELDS =
    SET
        ATNM$W_NUMBER    = [0, 0, 16, 0],
        ATNM$W_HIGH_ORD  = [0, 16, 16, 0],
        ATNM$V_PROTOCOL  = [0, 16, 11, 0],
        ATNM$V_FLAGS     = [0, 27, 5, 0],
        ATNM$V_SYSTEM    = [0, 27, 1, 0],
        ATNM$V_GLOBAL    = [0, 28, 1, 0],
        ATNM$V_CUSTOMER  = [0, 29, 1, 0]
    TES;
```

```
!+
!       Page Number Block (PNB)
!
!    31                              18 17                              0
!    +--------------------------------+--------------------------------+
!    |       Page Group Size          |          Page Number           |
!    +--------------------------------+--------------------------------+
!
!       The page number block define the break down of the page number.
!       Bits 0 to 17 is the page number.  Bits 18 to 31 define the number
!       of consecutive pages that can be read or written at one time.
!-

LITERAL
    PNB$S_BLOCK_LENGTH = 4;

MACRO
    $PNB = BLOCK[PNB$S_BLOCK_LENGTH, BYTE] FIELD (PNB$Z_FIELDS)
    %;

FIELD   PNB$Z_FIELDS =
    SET
        PNB$V_PAGE_NUM  = [0, 0, 17, 0],
        PNB$V_GROUP_SIZ = [0, 17, 15, 0]
    TES;
```

```
!+
!     Security Preservation Block (SPB)
!
!       3                   1
!       1                   5                   0
!      +---------+---------+---------+---------+  +0
!      |                                         |
!      |          Pointer to next                |      SPB$A_NEXT_SPB
!      |     Securtiy Preservation Block         |
!      +---------+---------+---------+---------+  +4
!      |                                         |
!      |          Pointer to prior               |      SPB$A_PRIOR_SPB
!      |     Securtiy Preservation Block         |
!      +---------+---------+---------+---------+  +8
!      |                                         |
!      |            Pointer to                   |      SPB$A_NCB
!      |          Node Control Block             |
!      +---------+---------+---------+---------+  +12
!      |                                         |
!      |          Granted Privileges             |      SPB$L_GRANT
!      |                                         |
!      +---------+---------+---------+---------+  +16
!      |                                         |
!      |          Denied Privileges              |      SPB$L_DENY
!      |                                         |
!      +---------+---------+---------+---------+  +20
!      |                                         |
!      |          Banished Privileges            |      SPB$L_BANISH
!      |                                         |
!      +---------+---------+---------+---------+  +24
!      |                                         |
!      |            Access Rights                |      SPB$L_ACCESS_RIGHTS
!      |                                         |
!      +---------+---------+---------+---------+  +28    SPB$V_PASSWORD
!      | Pswrd's | Pswrd's |    Entity's         |        SPB$W_PSW_LENGTH
!      | class   |  data   |    password         |        SPB$B_PSW_DTYPE
!      |         |  type   |     length          |        SPB$B_PSW_CLASS
!      +---------+---------+---------------------+  +32
!      |                                         |
!      |       Pointer to entity's               |      SPB$A_PSW_POINTER
!      |          password string                |
!      +-----------------------------------------+  +36
!                                | Must    |
!                                | Scan    |            SPB$B_SCAN_STATUS
!                                | Status  |
!                                +---------+  +37
!
!-

LITERAL
    SPB$S_BLOCK_LENGTH = 37;

MACRO
```

```
    $SPB = BLOCK[SPB$S_BLOCK_LENGTH, BYTE] FIELD (SPB$Z_FIELDS)
    %;

FIELD   SPB$Z_FIELDS =
    SET
        SPB$A_NEXT_SPB          = [0, 0, 32, 0],
        SPB$A_PRIOR_SPB         = [4, 0, 32, 0],
        SPB$A_NCB               = [8, 0, 32, 0],
        SPB$L_GRANT             = [12, 0, 32, 0],
        SPB$L_DENY              = [16, 0, 32, 0],
        SPB$L_BANISH            = [20, 0, 32, 0],
        SPB$L_ACCESS_RIGHTS     = [24, 0, 32, 0],
        SPB$V_PASSWORD          = [28, 0, 0, 0],
        SPB$W_PSW_LENGTH        = [28, 0, 16, 0],
        SPB$B_PSW_DTYPE         = [30, 0, 8, 0],
        SPB$B_PSW_CLASS         = [31, 0, 8, 0],
        SPB$A_PSW_POINTER       = [32, 0, 32, 0],
        SPB$B_SCAN_STATUS       = [36, 0, 8, 0]
    TES;

LITERAL
    SPB$K_QUE_HEADER            = BLOCK[0, SPB$A_NEXT_SPB; , BYTE];
```

```
    %SBTTL      'System Literal Definitions'
!++
!
!       SYSTEM LITERAL DEFINITIONS
!
!
!       These literals are only used internally.
!--

!+
!       CDD Implementation Version
!-

LITERAL
    CDD$K_FACILITY              = 43,
    CDD$K_LOWEST_VERSION        = 201,  ! Lowest compatible version
    CDD$K_VERSION               = 201;  ! Present version


!+
!       Boolean literals
!-

LITERAL
    TRUE    = 1,            ! Boolean TRUE value
    FALSE   = 0;            ! Boolean FALSE value


!+
!       The following literals are used to validate routines' parameter
!       lists.
!-

LITERAL
    ARG$K_OPTIONAL      = 1,    ! Parameter is optional
    ARG$K_REQUIRED      = 2,    ! Parameter is required
    ARG$K_SYNC          = 3,    ! See CDD$$U_VALIDATE documentation
    ARG$K_SYNCIF        = 4,    ! See CDD$$U_VALIDATE documentation
    ARG$K_MARK          = 0,    ! Mark parameter as missing
    ARG$K_DEFAULT       = 1,    ! Use default value
    ARG$K_STRING        = 1,    ! Default value is a null string
    ARG$K_LONG          = 2,    ! Default value is a longword
    ARG$K_WORD          = 3,    ! Default value is a word value
    ARG$K_REF           = 0,    ! Parameter passed by reference
    ARG$K_VALUE         = 1;    ! Parameter passed by value


!+
!       Access Lock Constants
!
!       NOTE:   The order of the LOCK$K_NORMAL lock constants MUST be
!               the same as the order of the CIB$V_REF_COUNTS lock
!               ref count fields in the CIB.
!-
```

```
LITERAL
    LOCK$K_NULL                 = 0,            ! Not locked

    LOCK$K_NORMAL               = 1,            ! Base of partially queued locks
        LOCK$K_PRESENCE         = 1,
        LOCK$K_RETRIEVAL        = 2,
        LOCK$K_UPDATE           = 3,
        LOCK$K_DELETE           = 4,
    LOCK$K_NORMAL_TYPES         = 4,            ! Number of partially queued locks

!!!     LOCK$K_QUEUED           = 5,            ! Base of fully queued locks
!!!     LOCK$K_FULL_RET         = 5,
!!!     LOCK$K_FULL_UPD         = 6,
!!!     LOCK$K_QUEUED_TYPES     = 2,            ! Number of fully queued locks

    LOCK$K_XXX_GET              = 1^17-1^16,    ! Flag indicates must establish lock
    LOCK$K_XXX_GET_IF           = 1^18-1^17,    ! Get lock if not already present
    LOCK$K_OPTIONS              = 1^32-1^16;    ! Option bits for locks


!+
!       Types of page purge requests
!-

LITERAL
    PURGE$K_ALL         = 1,            ! Complete purge
    PURGE$K_PRESENCE    = 2,            ! Purge but keep portal page
    PURGE$K_RETRIEVAL   = 3,            ! Checkpoint & keep retrieval locks
    PURGE$K_UPDATE      = 4,            ! Checkpoint & keep all locks
    PURGE$K_SUBTREE     = 100,          ! Purge whole subtree
    PURGE$K_CLUSTER     = 101;          ! Purge this cluster only


!+
!       Types of blocks that can be allocated in a pool.
!-

!+
!       Types of pools
!-

LITERAL
    MEM$K_LOWEST_POOL   = 1,            ! Lowest pool type
        MEM$K_CCB_POOL  = 1,            ! Pool for CCB and HCBs
        MEM$K_CIB_POOL  = 2,            ! Pool for cluster blocks
    MEM$K_HIGHEST_POOL  = 2;            ! Highest pool type


!+
!       CCB Pool block types
!-

LITERAL
    MEM$K_CCB_LOWEST            = 1,    ! Lowest block type in CCB pool
        MEM$K_CCB_CCB           = 1,    ! CCB block
```

```
        MEM$K_CCB_LOW_SLOT      = 2,    ! Lowest block type in free block list
            MEM$K_CCB_HCB       = 2,    ! HCB block
    MEM$K_CCB_HIGHEST           = 2;    ! Highest block type in CCB pool


!+
!       Cluster Pool block types
!-

LITERAL
    MEM$K_CIB_LOWEST            = 1,    ! Lowest block type in cluster pool
        MEM$K_CIB_CIB           = 1,    ! CIB block
        MEM$K_CIB_LOW_SLOT      = 2,    ! Lowest block type in free block list
            MEM$K_CIB_PCB       = 2,    ! PCB,
            MEM$K_CIB_LCB       = 3,    ! LCB,
            MEM$K_CIB_LCCB      = 4,    ! LCCB,
            MEM$K_CIB_ECCB      = 5,    ! ECCB,
            MEM$K_CIB_NCB       = 6,    ! NCB,
            MEM$K_CIB_SPB       = 7,    ! SPB,
    MEM$K_CIB_HIGHEST           = 7;    ! Highest block type in cluster pool

!+
!       These flags tell the deletion routine how it is to handle the
!       following cases:
!
!       DEL$K_FAST      says that pointers do not have to be cleaned up,
!                       as the block they reside in is going to be deleted.
!
!       DEL$K_PRESERVE  indicates that a directory node is merely to be
!                       emptied, and that its cluster is not to be deleted.
!
!       DEL$K_SUBDICTIONARY     says that sub-files are to have their contents
!                       deleted.
!-

LITERAL
    DEL$K_FAST          = 1^1 - 1^0,
    DEL$K_PRESERVE      = 1^2 - 1^1,
    DEL$K_SUBDICTIONARY = 1^3 - 1^2;


!+
!       User Identification Criteria
!-

LITERAL
    CDD$K_ACL_LOWEST    = 1,
    CDD$K_ACL_PASSWORD  = 1,    ! PASSWORD
    CDD$K_ACL_TERMINAL  = 2,    ! TERMINAL name or class
    CDD$K_ACL_UIC       = 3,    ! UIC
    CDD$K_ACL_USERNAME  = 4,    ! USERNAME
    CDD$K_ACL_HIGHEST   = 4;
```

```
    %SBTTL      'Security Masks'
!++
!       SECURITY MASKS
!--

!+
!       CDD security bits
!-

LITERAL
    CDD$K_PROT_C        = 1^1 - 1^0,    ! CONTROL access
    CDD$K_PROT_D        = 1^2 - 1^1,    ! LOCAL DELETE access
    CDD$K_PROT_G        = 1^3 - 1^2,    ! GLOBAL DELETE access
    CDD$K_PROT_H        = 1^4 - 1^3,    ! HISTORY list entry creation access
    CDD$K_PROT_P        = 1^5 - 1^4,    ! PASS THRU access
    CDD$K_PROT_S        = 1^6 - 1^5,    ! SEE (read) access
    CDD$K_PROT_U        = 1^7 - 1^6,    ! UPDATE terminal node access
    CDD$K_PROT_X        = 1^8 - 1^7,    ! EXTEND directory node access
    CDD$K_PROT_F        = 1^9 - 1^8,    ! FORWARDing directory creation allowed

!+
!       Macro-security values
!-

    CDD$K_PROT_ANY      = 1^9 - 1^0,
    CDD$K_PROT_DELETE   = CDD$K_PROT_D OR
                          CDD$K_PROT_G,
    CDD$K_PROT_EXTEND   = CDD$K_PROT_F OR
                          CDD$K_PROT_X,
    CDD$K_PROT_UPDATE   = CDD$K_PROT_C OR
                          CDD$K_PROT_D OR
                          CDD$K_PROT_G OR
                          CDD$K_PROT_H OR
                          CDD$K_PROT_U OR
                          CDD$K_PROT_X OR
                          CDD$K_PROT_F,

!+
!       Other processor security bits
!-

!+
!       VAX-11 Datatrieve
!-

    CDD$K_DTR_PROT_E    = 1^17 - 1^16,  ! EXTEND file
    CDD$K_DTR_PROT_R    = 1^18 - 1^17,  ! READ file
    CDD$K_DTR_PROT_M    = 1^19 - 1^18,  ! MODIFY file
    CDD$K_DTR_PROT_W    = 1^20 - 1^19;  ! WRITE file
```

```
    %SBTTL      'User Literal Definitions'
!+
!
!       USER LITERAL DEFINITIONS
!
!
!       These symbols are needed by users of the program interface.
!-


!+
!       System Defined Attribute Names
!-

LITERAL
    CDD$K_SYSNAM_FLAGS  = 1^28 OR 1^27 OR 0^16; ! Global/System-defined/Protocol=0

LITERAL
    CDD$K_FIRST_SYSNAM  = 1 OR CDD$K_SYSNAM_FLAGS,      ! Lowest system defined attribute name value
    CDD$K_FILE          = 1 OR CDD$K_SYSNAM_FLAGS,      ! Node's file name
    CDD$K_HISTORY       = 2 OR CDD$K_SYSNAM_FLAGS,      ! History list head
    CDD$K_NAME          = 3 OR CDD$K_SYSNAM_FLAGS,      ! Node's name
    CDD$K_PROTOCOL      = 5 OR CDD$K_SYSNAM_FLAGS,      ! Node's protocol name
    CDD$K_TYPE          = 6 OR CDD$K_SYSNAM_FLAGS,      ! Type of object pointed to by location code
    CDD$K_PATHNAME      = 7 OR CDD$K_SYSNAM_FLAGS,      ! Node's complete pathname
    CDD$K_SHORT_PATHNAME= 8 OR CDD$K_SYSNAM_FLAGS,      ! Node's path to CDD$DEFAULT directory
    CDD$K_ORDER         = 9 OR CDD$K_SYSNAM_FLAGS,      ! Directory's order
    CDD$K_LAST_SYSNAM   = 9 OR CDD$K_SYSNAM_FLAGS;      ! Highest system defined attribute name value


!+
!       Attribute and Entity Types
!-

LITERAL
    CDD$K_FIRST_TYPE    = 1,
    CDD$K_ENTITY        = 1,
    CDD$K_ENTITY_LIST   = 2,
    CDD$K_NULL          = 3,
    CDD$K_NUMERIC       = 4,
    CDD$K_STRING        = 5,
    CDD$K_STRING_LIST   = 6,
    CDD$K_DIRECTORY     = 7,
    CDD$K_TERMINAL      = 8,
    CDD$K_LAST_TYPE     = 8;


!+
!       User's entity purge options
!-

LITERAL
    CDD$K_ALL           = 1^1 - 1^0,
    CDD$K_ABORT         = 1^2 - 1^1,
    CDD$K_CHECKPOINT    = 1^3 - 1^2;
```

```
!+
!       User's node creation options
!-

LITERAL
    CDD$K_NOHISTORY     = 1^1 - 1^0,    ! Doesn't want history list cluster
    CDD$K_NOACL         = 1^2 - 1^1,    ! Don't create default ACL entry
    CDD$K_CREATE        = 1^3 - 1^2,    ! Create dictionary file if needed
    CDD$K_FIRST         = 1^4 - 1^3,    ! Insert as first node
    CDD$K_LAST          = 1^5 - 1^4;    ! Insert as last node


!+
!       User's node deletion options
!-

LITERAL
    CDD$K_CHECK         = 1^1 - 1^0,    ! Fail if directory has children
    CDD$K_SUBDICTIONARY = 1^2 - 1^1;    ! Delete contents of subdictionaries


!+
!       Values of the CDD$K_ORDER attribute
!-

LITERAL
    CDD$K_SORTED        = 1,            ! Directory is sorted
    CDD$K_NONSORTED     = 2;            ! Directory is not sorted
```

```
    %SBTTL      'LINKAGE DEFINITIONS'
!++
!
!       LINKAGE DEFINITIONS
!
!--

!+
!       CDDCALL
!
!       This linkage uses the CALLG/CALLS linkage convention, except
!       that it allows for one global register to be used in
!       parameter passing.
!
!       R11 -- used to pass the user's context pointer.
!-

LINKAGE
    CDDCALL = CALL : GLOBAL (USER_CONTEXT = 11);


!+
!       SYS_JSB
!
!       This linkage provides us with a general JSB routine linkage.
!-

LINKAGE
    SYS_JSB = JSB;
```

```
    %SBTTL      'MACRO DEFINITIONS'
!++
!
!       MACRO DEFINITIONS
!
!--


!+
!       $ACTIVE
!       $INACTIVE
!
!       These macroes declare that we have started, and finished,
!       respectively, a CDD transaction.  They abort the transaction
!       if another transaction is in progress.
!-

MACRO
    $ACTIVE =
        BEGIN
            EXTERNAL
                CDD$GB_INUSE:           BYTE;

            EXTERNAL LITERAL
                CDD$_NOTASTREE;

            BUILTIN
                TESTBITSS;


            IF TESTBITSS (CDD$GB_INUSE) THEN
                SIGNAL (CDD$_NOTASTREE);
        END
    %,

    $INACTIVE =
        BEGIN
            EXTERNAL
                CDD$GB_INUSE:           BYTE;

            CDD$GB_INUSE = FALSE;
        END
    %,


!+
!       $BITCLEAR
!
!       This macro checks to see if any bit in a mask is set in the
!       target area.  If not, it returns TRUE.
!-

    $BITCLEAR(target, mask) =
        (target AND mask) EQLU 0
    %,
```

```
!+
!       $BITSET
!
!       This macro checks to see if any bit in a mask is set in the
!       target area.  If so, it returns TRUE.
!-

    $BITSET(target, mask) =
        (target AND mask) NEQU 0
    %,


!+
!       $DONE_TRANS
!
!       This macro is used to terminate a transaction.
!
!       Call:
!
!           $DONE_TRANS [(dsc1 [, dsc2] ...)]
!
!       Where:
!
!           dsci :== the names of dynamic descriptors which are to
!                    have their strings returned to the string pool.
!-

    $DONE_TRANS (dsc1) =
        BEGIN
            EXTERNAL ROUTINE
                CDD$$SN_DONE_TRANS      : CDDCALL       NOVALUE;


            CDD$$SN_DONE_TRANS (dsc1
                %IF NOT %NULL(%REMAINING) %THEN , %REMAINING %FI );
            .USER_CONTEXT[CCB$L_STATUS]
        END
    %,


!+
!       $FIND_ENTITY
!
!       This macro returns the virtual address of the LCCB associated
!       with a location code.  It also checks to make certain that
!       the cluster is locked as requested, and that the cluster's node
!       allows the requested security access.
!
!       Call:
!
!           lccb-block.wa.v = $FIND_ENTITY (valid-arg.ra.v ,
!                                 RETRIEVAL           READ
!               (  CHECK      { UPDATE      } ) , { MODIFY  } );
!                               DELETE            DELETE
```

```
!                                                   ANY
!
!       Where:
!
!           valid-arg :== the address of the calling routine's validated
!                         argument list.
!                         The argument list must have the following format:
!
!                               valid-arg[0]  ::= address of longword holding
!                                                 context #
!                               valid-arg[1]  ::= address of descriptor holding
!                                                 path name, or zero.
!
!           The first set of keywords names the desired lock state of
!           the entity's cluster.
!
!           The last set of keywords names the intended access to the
!           cluster.
!-

    $FIND_ENTITY (valid_arg, locking, security) =
        BEGIN
        EXTERNAL ROUTINE
            CDD$$SN_FIND_ENTITY     : CDDCALL;

        CDD$$SN_FIND_ENTITY (..valid_arg[1],
            $FIND_XXX_LOCKING (%REMOVE(locking)),
            %NAME('CDD$K_PROT_', security))
        END
    %,

    $FIND_XXX_LOCKING (class, type) =
        %IF %IDENTICAL (class, %QUOTE LOCK) %THEN
            LOCK$K_XXX_GET OR
        %ELSE
            %IF %IDENTICAL (class, %QUOTE LOCKIF) %THEN
                LOCK$K_XXX_GET_IF OR
            %ELSE
                %IF NOT %IDENTICAL (class, %QUOTE CHECK) %THEN
                    %ERROR ('Invalid locking keyword: ', class)
                %FI
            %FI
        %FI

        %NAME ('LOCK$K_', type)
    %,


!+
!       $FIND_NODE
!
!       This macro returns the virtual address of the NCB associated
!       with a path name or location code.  It also checks to make certain
!       that the cluster is locked as requested, and that the target node
!       allows the requested security access.
```

```
!       Call:
!
!           ncb-block.wa.v = $FIND_NODE (valid-arg.ra.v ,
!                   LOCK        RETRIEVAL           READ
!               ( { LOCKIF } { UPDATE     } ) , { MODIFY  } );
!                   CHECK       DELETE              DELETE
!                                                   ANY
!
!       Where:
!
!           valid-arg :== the address of the calling routine's validated
!                         argument list.
!                         The argument list must have the following format:
!
!                               valid-arg[0]  ::= address of longword holding
!                                                 context #
!                               valid-arg[1]  ::= address of descriptor holding
!                                                 path name, or zero.
!                               valid-arg[2]  ::= address of longword holding
!                                                 location code, or zero.
!
!           The first set of keywords names the desired lock state of
!           the node's cluster.
!
!           The last set of keywords names the intended access to the
!           cluster.
!-

    $FIND_NODE (valid_arg, locking, security) =
        BEGIN
            EXTERNAL ROUTINE
                CDD$$SN_FIND_NODE       : CDDCALL;

            CDD$$SN_FIND_NODE (valid_arg,
                $FIND_XXX_LOCKING (%REMOVE(locking)),
                %NAME('CDD$K_PROT_', security))
        END
    %,

!+
!       $FIND_PARENT
!
!       This macro returns the virtual address of the NCB associated
!       with a location code.  It also checks to make certain that
!       the cluster is locked as requested, and that the cluster's node
!       allows the requested security access.
!
!       Call:
!
!           status.wlc.v = $FIND_PARENT (valid-arg.ra.v ,
!                   LOCK        RETRIEVAL           READ
!               ( { LOCKIF } { UPDATE     } ) , { MODIFY  } , ncb-block.wa.r,
!                   CHECK       DELETE              DELETE
!                                                   ANY
!               name.wt.ds);
!
```

```
!       Where:
!
!           valid-arg :== the address of the calling routine's validated
!                         argument list.
!                         The argument list must have the following format:
!
!                               valid-arg[0]  ::= address of longword holding
!                                                 context #
!                               valid-arg[1]  ::= address of descriptor holding
!                                                 path name, or zero.
!                               valid-arg[2]  ::= address of longword holding
!                                                 location code, or zero.
!
!           The first set of keywords names the desired lock state of
!           the target cluster.
!
!           The last set of keywords names the intended access to the
!           cluster.
!-

    $FIND_PARENT (valid_arg, locking, security, ncb_block, name) =
        BEGIN
            EXTERNAL ROUTINE
                CDD$$SN_FIND_PARENT     : CDDCALL;

            CDD$$SN_FIND_PARENT (valid_arg,
                $FIND_XXX_LOCKING (%REMOVE(locking)),
                %NAME('CDD$K_PROT_', security), ncb_block, name)
        END
    %,

!+
!       $INIT_DSC
!
!       This macro is used to initialize dynamic string descriptors.
!
!       Call:
!
!           $INIT_DSC (dsc1 [, dsc2] ...)
!-

    $INIT_DSC[DSC_NAM] =
        BEGIN
            DSC_NAM[DSC$B_DTYPE] = DSC$K_DTYPE_T;
            DSC_NAM[DSC$B_CLASS] = DSC$K_CLASS_D;
            DSC_NAM[DSC$W_LENGTH] = 0;
            DSC_NAM[DSC$A_POINTER] = 0;
        END
    %,


!+
!       $IO_SYNC (ef, iosb)
```

```
!++
!       This macro waits for the event flag (ef) to be set and for
!       the I/O status parameter (iosb) to be filled in (non-zero).
!
!       ef      must be the target event flag number.
!
!       iosb    must be the address of the I/O status block.  This
!               must be defined as a VECTOR[,WORD] VOLATILE structure.
!--

    $IO_SYNC (ef, iosb) =
        DO
            BEGIN
                LOCAL
                    STATUS:             LONG;

                STATUS = $WAITFR (efn = ef);
                IF NOT .STATUS THEN
                    SIGNAL_STOP (.STATUS);
                STATUS = $CLREF (efn = ef);
                IF NOT .STATUS THEN
                    SIGNAL_STOP (.STATUS);
            END WHILE .iosb[0] EQLU 0
    %,


!++
!       $MARK_PAGE (page-block)
!
!       This macro marks a page as modified.  Such a page must be
!       written back to the dictionary file when it is purged from
!       the staging buffers.
!--

    $MARK_PAGE (page_block) =
        page_block[PCB$V_MODIFIED] = TRUE
    %,


!++
!       $PARAMETERS (arg1 [, arg2] ...)
!
!       This macro is used to build the control vector for the parameter
!       list validate routine (CDD$$U_VALIDATE).
!
!       There is one entry (arg1, arg2, etc) for every formal parameter in
!       the routine.  Each entry has the following format:
!
!       ( REQUIRED , [n] , [REF ! VALUE] !
!         { SYNC ! SYNCIF }, n    !
!         OPTIONAL [,MARK ! ,DEFAULT [,STRING ! ,LONG ! ,WORD] ] )
!--

    $PARAMETERS[] =
        UPLIT (%LENGTH, $PARM_PROCESS (%REMOVE(%REMAINING)))
```

```
    %,

    $PARM_PROCESS[arg] =
        BYTE ($PARM_DECIDE (%REMOVE(arg)))
    %,

    $PARM_DECIDE(status)[] =
        %IF %IDENTICAL (status, %QUOTE REQUIRED) %THEN
            ARG$K_REQUIRED, 0,
            %IF %NULL (%REMAINING) %THEN
                ARG$K_REF, 0
            %ELSE
                $PARM_REQUIRED (%REMAINING)
            %FI
        %ELSE
            %IF %IDENTICAL (status, %QUOTE SYNC) %THEN
                ARG$K_SYNC, 0, 0, %REMAINING
            %ELSE
                %IF %IDENTICAL (status, %QUOTE SYNCIF) %THEN
                    ARG$K_SYNCIF, 0, 0, %REMAINING
                %ELSE
                    %IF %IDENTICAL (status, %QUOTE OPTIONAL) %THEN
                        ARG$K_OPTIONAL, $PARM_OPTIONAL (%REMAINING), 0
                    %ELSE
                        %ERROR ('Invalid parameter status: ', status)
                        0, 0, 0, 0
                    %FI
                %FI
            %FI
        %FI
    %,

    $PARM_REQUIRED(number, pass) =
        %IF %NULL (pass) %THEN
            0
        %ELSE
            %NAME ('ARG$K_', pass)
        %FI
        %IF %NULL (number) %THEN
            0
        %ELSE
            , number
        %FI
    %,

    $PARM_OPTIONAL(action)[] =
        %IF %IDENTICAL (action, %QUOTE MARK) %THEN
            ARG$K_MARK, 0
        %ELSE
            %IF %IDENTICAL (action, %QUOTE DEFAULT) %THEN
                ARG$K_DEFAULT, %NAME ('ARG$K_', %REMAINING)
            %ELSE
                %ERROR ('Invalid action keyword: ', action)
                0, 0
            %FI
        %FI
```

```
    %,


!+
!       $PRESENT
!
!       This macro checks to see if a parameter is present (non-zero)
!       in a list.
!-
    $PRESENT (name) =
        .name NEQA 0
    %,


!+
!       $RECOVERY ( RESET ! ENABLE ! DISABLE )
!
!       This macro determines the ability of the exit handler to
!       perform a purge of the cache if the task aborts.
!
!       $RECOVERY (DISABLE)
!                       declares that the internal data structures or
!                       external disk structure is in an indeterminant
!                       state and cannot be recovered by the exit
!                       handler.
!
!       $RECOVERY (ENABLE)
!                       declares that the data structure manipulation
!                       is completed.
!
!       $RECOVERY (RESET)
!                       specifies that the data structures are in a
!                       recoverable state.
!
!       Note that these can be nested.  Recovery is only possible if
!       the recovery counter is zero (reset).
!-

    $RECOVERY (option) =
        BEGIN
            EXTERNAL
                CDD$GW_RECOVERY:        WORD;

            %IF %IDENTICAL (option, %QUOTE DISABLE) %THEN
                CDD$GW_RECOVERY = .CDD$GW_RECOVERY + 1;
            %ELSE
                %IF %IDENTICAL (option, %QUOTE ENABLE) %THEN
                    CDD$GW_RECOVERY = .CDD$GW_RECOVERY - 1;
                %ELSE
                    %IF %IDENTICAL (option, %QUOTE RESET) %THEN
                        CDD$GW_RECOVERY = 0;
                    %ELSE
                        %ERROR ('Illegal recovery option: ', option)
                    %FI
                %FI
```

```
        %FI
    END
%,


!+
    $RELEASE_LOCK

    This macro calls the CDD$$SN_RELEASE routine to release one
    or more locks.

    Call:

                            RETRIEVAL
        $RELEASE_LOCK ( { UPDATE } , ncb-block1 ...);
                            DELETE
!-

$RELEASE_LOCK (locking)[] =
    BEGIN
        EXTERNAL ROUTINE
            CDD$$SN_RELEASE          : CDDCALL       NOVALUE;

        CDD$$SN_RELEASE (%NAME ('LOCK$K_', locking), %REMAINING)
    END
%,


!+
    $SIGNAL_SEVERE (error)

    This routine signals a severe error.
!-

$SIGNAL_SEVERE (ERROR) =
    SIGNAL (ERROR OR STS$K_SEVERE)
%,


!+
    $STATIC_DSC

    This macro is used to initialize static string descriptors.


    Call:

        $STATIC_DSC (dsc1 [, dsc2] ...)

    Where:

        dsci :== name ! ( name , source )

    Source is the name of another descriptor.  The named descriptor
    is initialized to point to the same string as source.
```

```
!
!-

    $STATIC_DSC[dsci] =
        $STATIC_DSC_BUILD (%REMOVE (dsci))
    %,

    $STATIC_DSC_BUILD(name, source) =
        BEGIN
            name[DSC$B_DTYPE] = DSC$K_DTYPE_T;
            name[DSC$B_CLASS] = DSC$K_CLASS_S;
            %IF %NULL(source) %THEN
                name[DSC$W_LENGTH] = 0;
                name[DSC$A_POINTER] = 0;
            %ELSE
                name[DSC$W_LENGTH] = .source[DSC$W_LENGTH];
                name[DSC$A_POINTER] = .source[DSC$A_POINTER];
            %FI
        END
    %,


!+
!       $STRING
!
!       These macros are used to build string descriptors for literal
!       strings.
!
!       Call:
!
!               $STRING ( (name, string) ...)
!               $STRING_INIT ();
!
!       "name" is defined to be a BLOCK structure, and the address of
!       the string is poked into the structure by the STRING_INIT macro,
!       which must be the first executable statement in a routine.
!-

    $STRING [] =
        $STRING_DSC_SETUP (%REMAINING)
        MACRO
            %QUOTE %QUOTE $STRING_INIT =
                $STRING_PTR_SETUP (%QUOTE %EXPAND %REMAINING)
            %QUOTE %
    %,

    $STRING_DSC_SETUP[PAIR] =
        $STRING_DSC_INIT (%REMOVE(PAIR))
    %,

    $STRING_DSC_INIT (STR_NAME, STR_VAL) =
        OWN
            STR_NAME:   $DSC PRESET ([DSC$B_DTYPE] = DSC$K_DTYPE_T,
                                     [DSC$B_CLASS] = DSC$K_CLASS_S,
                                     [DSC$W_LENGTH] =
                                         %CHARCOUNT (%REMOVE(STR_VAL)));
```

```
    %,

    $STRING_PTR_SETUP[PAIR] =
        $STRING_PTR_INIT (%REMOVE(PAIR))
    %,

    $STRING_PTR_INIT (STR_NAME, STR_VAL) =
        STR_NAME[DSC$A_POINTER] = UPLIT BYTE(%REMOVE(STR_VAL));
    %,



!+
!       $TEXTC
!
!       This macro is used to define counted strings.  The first byte
!       of such strings is a count of the number of characters in the
!       string.
!
!       Each string is defined to be a VECTOR[,BYTE] structure, with
!       the 0 element being the character count, and the actual string
!       starting at STRING[1].
!
!       $TEXTC ((name1,'str1') [, (name2, 'str2')] ...);
!-

    $TEXTC[PAIR] =
        $TEXTC_STR(%REMOVE(PAIR))
    %,

    $TEXTC_STR(NAME,TSTR) =
        BIND
            NAME = UPLIT BYTE (%CHARCOUNT(%REMOVE(TSTR)), %REMOVE(TSTR)) :
                VECTOR[%CHARCOUNT(%REMOVE(TSTR))+1, BYTE];
    %,



!+
!       $VALIDATE (cntl, [arg-list])
!
!       This macro generates a call to the general transaction setup
!       routine.
!
!       cntl    -- the address of the control vector for CDD$$U_VALIDATE.
!
!       arg-list-- is optional.  If present, it is the address of the
!                  vector which is to receive the verified argument list
!                  from CDD$$U_VALIDATE.
!-

    $VALIDATE(cntl, arg_list) =
        BEGIN
            EXTERNAL ROUTINE
                CDD$$SN_START_TRANS     : CDDCALL NOVALUE;
```

```
            BUILTIN
                AP;


            %IF %NULL (arg_list) %THEN
                CDD$$SN_START_TRANS (.AP, cntl)
            %ELSE
                CDD$$SN_START_TRANS (.AP, cntl, arg_list)
            %FI
        END
    %;
```

CDDENTRY
LIS

CDD

CDDSHR
MAP

CDDEXC2
LIS

CDDLIB
LIS

CDDEXC
LIS

WRITEBOOT
LIS

CDDLIB
B32